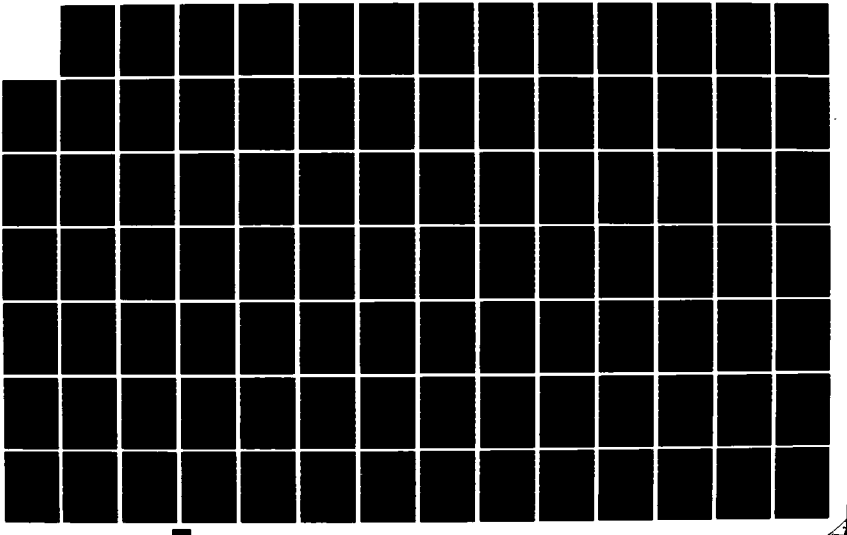
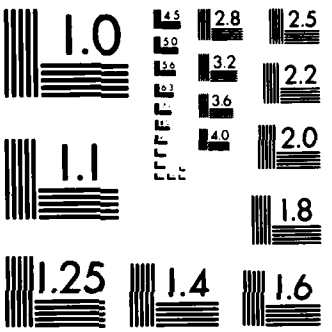
AD-A151 622 CW MEASUREMENT SYSTEM SOFTWARE SYSTEM MAINTENANCE MANUAL(U) EG AND G WASHINGTON ANALYTICAL SERVICES CENTER INC ALBUQUERQU... R NELSON ET AL. 02 APR 82 1/3

UNCLASSIFIED EG/G-AG-1435 DNA-6232F DNA001-80-C-0290 NL





(2)

DNA 6232F

CW MEASUREMENT SYSTEM
Software System Maintenance Manual

EG&G Washington Analytical Services Center, Inc.
2450 Alamo Avenue SE
Albuquerque, New Mexico 87106

2 April 1982

Final Report for Period 27 May 1980 —2 April 1982

CONTRACT No. DNA 001-80-C-0290

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED.

THIS WORK WAS SPONSORED BY THE DEFENSE NUCLEAR AGENCY
UNDER RDT&E RMSS CODE B362080462 G52AAXEX40502 H2590D.

Prepared for
Director
DEFENSE NUCLEAR AGENCY
Washington, DC 20305

Destroy this report when it is no longer needed. Do not return to sender.

PLEASE NOTIFY THE DEFENSE NUCLEAR AGENCY, ATTN: STTI, WASHINGTON, D.C. 20305, IF YOUR ADDRESS IS INCORRECT, IF YOU WISH TO BE DELETED FROM THE DISTRIBUTION LIST, OR IF THE ADDRESSEE IS NO LONGER EMPLOYED BY YOUR ORGANIZATION.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER DNA 6232F	2. GOVT ACCESSION NO. AD-A115 632	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* CW MEASUREMENT SYSTEM Software System Maintenance Manual		5. TYPE OF REPORT & PERIOD COVERED Final Report for Period 27 May 80—2 Apr 82
		6. PERFORMING ORG. REPORT NUMBER AG-1435
7. AUTHOR(s) Rick Nelson Pat Lindsey		8. CONTRACT OR GRANT NUMBER(s) DNA 001-80-C-0290
9. PERFORMING ORGANIZATION NAME AND ADDRESS EG&G Washington Analytical Services Center, Inc. 2450 Alamo Avenue, SE Albuquerque, New Mexico 87106		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Task G52AAXEX-40502
11. CONTROLLING OFFICE NAME AND ADDRESS Director Defense Nuclear Agency Washington, DC 20305		12. REPORT DATE 2 April 1982 13. NUMBER OF PAGES 214
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS *(of this report)* UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A since UNCLASSIFIED

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

This work was sponsored by the Defense Nuclear Agency under RDT&E RMSS Code B362080462 G52AAXEX40502 H2590D.

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

PDP-11	Flecs	FORTRAN	Pseudo-Device
Software	PCU	Message	
Firmware	CW	GPIB	
Structured Programming	Data Structure	IEEE 488 Bus	
EPROM	Global Flag	Logical Device	

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Manual for use by system engineers for maintenance of CWMS Upgrade software system. Covers design details of PDP-11 software, and INTEL 8080 PCU software, including system data structures, communication protocol, software module descriptions, and description of coding practices and languages used to generate the software.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

CONVERSION FACTORS FOR U.S. CUSTOMARY TO METRIC (SI) UNITS OF MEASUREMENT

To Convert From	To	Multiply By
angstrom	meters (m)	1.000 000 X E -10
atmosphere (normal)	kilo pascal (kPa)	1.013 25 X E +2
bar	kilo pascal (kPa)	1.000 000 X E +2
barn	meter^2 (m^2)	1.000 000 X E -28
British thermal unit (thermochemical)	joule (J)	1.054 350 X E +3
cal (thermochemical)/cm^2§	mega joule/m^2 (MJ/m^2)	4.184 000 X E -2
calorie (thermochemical)§	joule (J)	4.184 000
calorie (thermochemical)/g§	joule per kilogram (J/kg)*	4.184 000 X E +3
curie§	giga becquerel (GBq)+	3.700 000 X E +1
degree Celsius§	degree kelvin (K)	$t_K = t°_C + 273.15$
degree (angle)	radian (rad)	1.745 329 X E -2
degree Fahrenheit	degree kelvin (K)	$t_K = (t°_F + 459.67)/1.8$
electron volt§	joule (J)	1.602 19 X E -19
erg§	joule (J)	1.000 000 X E -7
erg/second	watt (W)	1.000 000 X E -7
foot	meter (m)	3.048 000 X E -1
foot-pound-force	joule (J)	1.355 818
gallon (U.S. liquid)	meter^3 (m^3)	3.785 412 X E -3
inch	meter (m)	2.540 000 X E -2
jerk	joule (J)	1.000 000 X E +9
joule/kilogram (J/kg) (radiation dose absorbed)§	gray (Gy)*	1.000 000
kilotons§	terajoules	4.183
kip (1000 lbf)	newton (N)	4.448 222 X E +3
kip/inch^2 (ksi)	kilo pascal (kPa)	6.894 757 X E +3
ktap	newton-second/m^2 (N-s/m^2)	1.000 000 X E +2
micron	meter (m)	1.000 000 X E -6
mil	meter (m)	2.540 000 X E -5
mile (international)	meter (m)	1.609 344 X E +3
ounce	kilogram (kg)	2.834 952 X E -2
pound-force (lbf avoirdupois)	newton (N)	4.448 222
pound-force inch	newton-meter (N·m)	1.129 848 X E -1
pound-force/inch	newton/meter (N/m)	1.751 268 X E +2
pound-force/foot^2	kilo pascal (kPa)	4.788 026 X E -2
pound-force/inch^2 (psi)	kilo pascal (kPa)	6.894 757
pound-mass (lbm avoirdupois)	kilogram (kg)	4.535 924 X E -1
pound-mass-foot^2 (moment of inertia)	kilogram-meter^2 ($kg·m^2$)	4.214 011 X E -2
pound-mass/foot^3	kilogram-meter^3 (kg/m^3)	1.601 846 X E +1
rad (radiation dose absorbed)§	gray (Gy)*	1.000 000 X E -2
roentgen§	coulomb/kilogram (C/kg)	2.579 760 X E -4
shake	second (s)	1.000 000 X E -8
slug	kilogram (kg)	1.459 390 X E +1
torr (mm Hg, 0° C)	kilo pascal (kPa)	1.333 22 X E -1

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

LIST OF ILLUSTRATIONS

SECTION 1
SYSTEM OVERVIEW

The CW Measurement Data Acquisition System is a real-time, multi-task, event driven software system using the Digital Equipment Corp's RSX-11M real-time operating system and software produced by EG&G WASC Albuquerque Operation Systems Group. The operating system is documented by DEC in manuals supplied with the system. This document describes the application software written by EG&G.

The CW Measurement System consists of two basic sections: the cw generation and measurement subsystem built by Boeing and modified by EG&G, and the Data Acquisition subsystem consisting of a DEC PDP-11/34A, two RL01 disk drives, an HP-2648A operator's console and other associated hardware. The data detected by the measurement subsystem receiver are transmitted to the PDP-11 via an RS-232 data link as 36-byte data records. These data are then processed in real-time; the instrumentation and sensor effects are corrected, the data are displayed on the operator's console and, optionally, saved on cassette or disk for future processing and archival storage.

The software system includes the following:

- Data input task, which receives data from the measurement subsystem and sends it to the corrections task.
- Corrections and buffering task, which spools the raw data received from the input task. As data are received and time allows, the data are despooled, sensor and instrumentation corrections are applied, and the data are dispatched to the other system tasks.
- Operator Console Monitor task (CRT), which handles the interface between the operator and the system. It prompts the operator for information concerning the test, and controls the graphic display.

- Inverse Fourier transform task, which converts data from the frequency domain into the time domain, applying the operator-selected Butterworth filter threat waveform in the process.
- Hard-Copy plot task, which plots the corrected data on the flatbed digital plotter, if requested.
- Tape task, which writes corrected and transformed data to tape for storage and/or future processing, if requested.
- Disk-Save task, which writes the requested data domains to the disk for storage for future processing.
- Setup the remote data links task, which controls the optical data links and VHF switch.
- Name the mission file task, which allows the operator to request an old mission file or create a new one.

These tasks communicate with each other by event flags and system messages. Each task processes data as it arrives, at its own speed, asynchronously. Figure 1-1 represents the task architecture.

The CW Measurement Data Acquisition System includes a number of programs which may be run either offline or concurrently with data acquisition if a second terminal is available. These programs form the Interactive Analyst Package and enable a data analyst to perform mathematical operations on data, to plot, list, scale and transform data, to edit data files, to list the contents of file headers, to add headers to data files, to list the contents of the mission file and to initialize tapes, copy data from tape to disk and disk to tape.

The system also contains a number of stand-alone utility programs for use in supporting the various facilities of the system. These include:

- Analytic probe calibration task. This task generates a probe calibration file for B-dot type sensors.

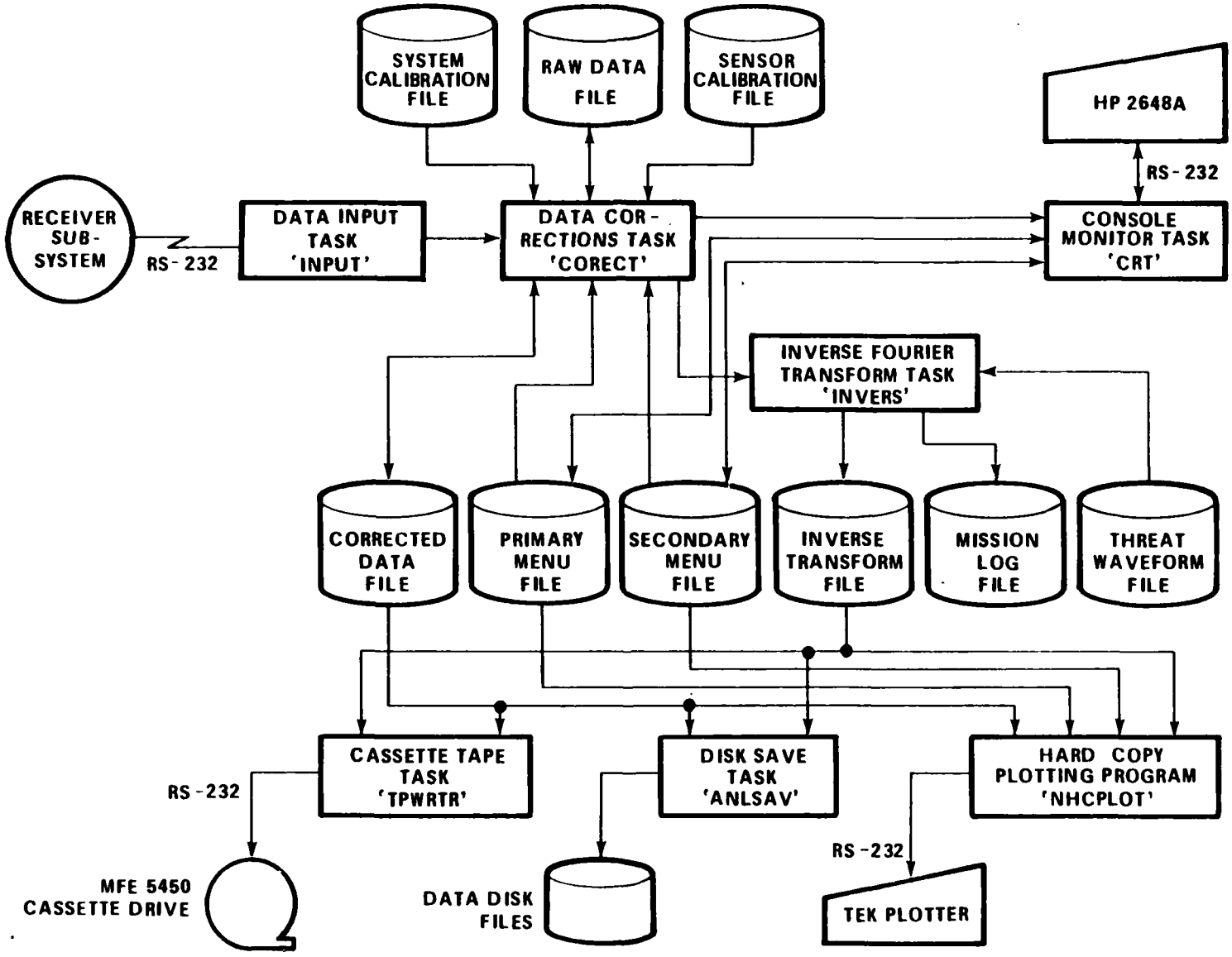


Figure 1. Software System Flow Diagram

- Threat waveform task. This task generates threat waveform files. The values for α and β describing the threat waveform and the high-frequency and low-frequency cutoff points of a ninth order Butterworth filter are specified by the operator.
- Delete frequency EPROM task. Burns EPROMs which contain frequencies to be deleted from the CW spectrum during measurements.
- Amplitude control EPROM task. Burns EPROMs which control the power output of the transmitter power amplifier during a test.

The design of these tasks is described by Program Design Language (PDLs). PDLs are described fully in Section 5 of this manual. The PDLs for these tasks are contained in Section 1 of the Listings Manual.

Section 2 of this manual contains a description of all the tasks in the data collection system, along with details of data structures used by each task, and the files generated or used by each. Section 3 describes the Interactive Analyst Package software. Section 4 describes the system data structures and global variables. Section 5 describes the software development tools used and Section 6 describes the RSX-11M environment and support programs. Sections 7 and 8 relate to the Boeing Continuous Wave Measurement Subsystem (CWMS). Section 7 describes the function of the Program Control Unit (CPU) and Section 8 describes the software modules in the PCU firmware.

SECTION 2
DATA COLLECTION MODULES

2.1 INPUT TASK ('INPUT')

The data input task, INPUT, receives data from the receiver program control unit (or the MFE tape unit) and sends the data to the corrections task, CORECT, for spooling and distribution. This task is small and fast in comparison to other tasks within the system. Since it is required that the task have "immediate" turnaround of a received data record, it operates at the highest priority of all tasks in the software subsystem. This allows INPUT to process the data from the receiver PCU as soon as it arrives and to immediately start another read, insuring that no data are lost.

The task issues RSX-11M QIO directives to read the data from the PCU. INPUT maintains two data buffers. When a record from the PCU is received, control is passed to INPUT by the RSX-11M system. INPUT then starts the read-and-proceed QIO directive with the inactive buffer, and processes the data in the active buffer (the buffer which the last read accessed).

When a data record arrives, the first byte of the record is read to determine the record type. (See paragrah 4-3 for the particulars of the PCU record formats.) Data are then extracted from the record and placed into messages which are sent to CORECT via the RSX-11M Send Message directive. There are three types of PCU records. One type is the Panel Data Block (PDB) record containing information about the receiver PCU front panel switch settings. The second record type is the Data record containing raw data for spooling and reduction. The third is the Error Status Block record containing an error code for an error sensed by the receiver PCU. Each record type is formatted differently. INPUT extracts the information and formats a message (or multiple messages when the record contains more data than can be packed into a single message) to send to CORECT. If this is completed before the read-and-proceed QIO

issued earlier, the task waits for the read to complete, freeing the PDP-11 so other tasks can execute.

When the PCU sends an end-of-sweep Data record, indicating the completion of a cycle, INPUT decrements a counter which was initially set to the number of measurement cycles contained in the panel data block. When the counter reaches 0, the task goes to end-of-task. This allows INPUT to read data which were previously recorded on cassette in the 'secondary' configuration. If the tape cassette was previously written upon, it may have extraneous data following the measurement. The shut-off feature insures that this data are not read into the system.

2-2 CORRECTIONS AND BUFFERING TASK ('CORECT')

CORECT accepts raw data from the input task ('INPUT') and writes it to disk. The raw data are then read and processed asynchronously with the other tasks. When the data are read, CORECT applies corrections to it and spools the corrected data; then dispatches the corrected data to the CRT task for plotting and to the Inverse Transform task. CORECT controls up to seven disk files which contain system and probe calibration data, raw data and corrected data.

Internally, the task is in five parts. Each part is invoked by sensing an event flag, set either by another task or by CORECT itself. The task waits for one of these flags to be set, then enters the appropriate section of code. The flags signal one of the following events:

- Correction Data Available — the raw data input task has sent a data item to CORECT.
- CRT Waiting for Data — CRT is waiting for corrected data.
- Inverse Waiting for Data — the Inverse Transform task is waiting for corrected data.

- Primary Menu Ready — CRT is signaling that the primary menu entries are correct.
- Raw Data Waiting for Correction — there is data in the raw data file ready for corrections (this flag is controlled by CORECT itself).

When corrected data are requested by CRT or INVERS, CORECT reads the next data point from the corrections file and dispatches it to the requester using the Send Message facility of RSX-11M, and signals the requester by setting a global event flag. When the INPUT task signals it has raw data to correct, CORECT invokes the Receive Message facility of RSX-11M and writes the data. The Primary Menu Ready flag causes CORECT to open the primary menu file, extract the information it needs and close the file. The Raw Data Waiting for Correction flag is set when raw data are received from INPUT, and is reset when the raw data file is empty. An abort flag is also used to signal error conditions from which there is no recovery. Setting the abort flag causes all tasks in the system to go to end-of-task immediately.

The files used by CORECT are all kept in UFD [200,1]. (See the RSX-11M System Documentation Manual 1A, the section entitled 'EXECUTIVE' for a discussion of UFDs and UICs, also refer to paragraph 4-2.1.1.) Volume SY: contains all system and sensor calibration files, the menu files, and a composite correction file built whenever a multicycle test is performed to save computation time. Volume CD: (the Classified data disk; normally mounted on DL1:) contains the raw data file and the corrected data file. All files are direct access, unformatted files. Except for the menu files, which are discussed in the next section, and the inverse transform file, all files contain 12 byte records. These records are subdivided into three fields which contain the frequency, amplitude (in dB), and phase (in degrees) respectively in internal single precision floating

point format. For measurements which do not contain phase information, the phase field is set to 0.

The names of the files used by CORECT are:

- MENU.PRI — primary menu file
- CORECT.DAT — corrected data file
- RAWDATA.TMP — input spool file
- ACOMP.TMP — composite correct file for use in multicycle tests
- SYSTF.CAL — system calibration file for transfer function measurements
- SYSRF.CAL — system calibration file for response function measurements
- XXXXXX.CAL — sensor calibration file where XXXXXX is the name (up to nine characters) of the sensor as entered in the primary menu

The equations for the corrections are derived from the previous cw data reduction work done by EG&G. These equations assume a measurement system similar to that used here.

The equation used for amplitude correction is:

$$\text{Corrected Ampl.} = K_a(S_{meas} - R_{meas}) + \text{REFGAIN} - \text{SIGGAIN} + \text{RPROBE} - \text{SPROBE} + \text{NADR} - \text{SCAL} \quad (1)$$

where

K_a = conversion factor from millivolts to dB for the network analyzer (here, $K_a = .02$)

$S_{meas}-R_{meas}$ = transfer function in millivolts of the signals sensed at the reference and the signal channels. In effect, this difference is the raw data received by the corrections task.

REFGAIN = Signed gain added to the reference channel.

SIGGAIN = Signed gain added to the signal channel.

RPROBE = reference sensor transfer function.

SPROBE = signal probe transfer function.

NADR = network analyzer display reference in dB (set on front panel of network analyzer).

SCAL = transfer function of system instrumentation exclusive of sensors (probes) and externally introduced gains or attenuations (i.e., the system cal).

This equation is derived from examination of the system setup as diagramed in Figure 2-1.

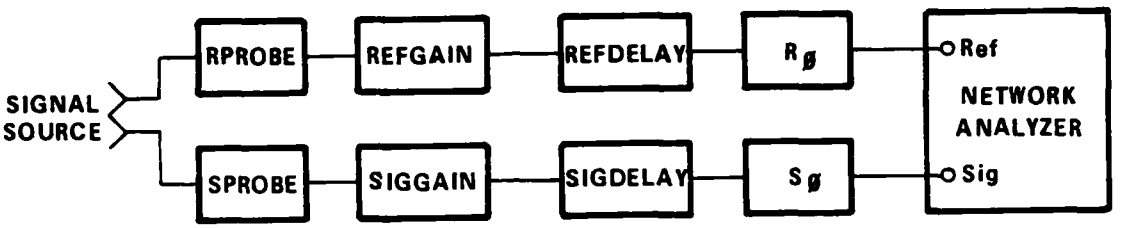


Figure 2-1. Test Network Configuration

The output of the network analyzer in dB is:

$$K_a(S_{meas} - R_{meas}) = S + SPROBE + SIGGAIN + SIGDELAY + S_\phi - R + RPROBE - REFGAIN - REFDELAY - R_\phi \quad (2)$$

where S is the signal at the signal source into the signal channel, R is the signal at the signal source into the reference channel. SIGDELAY and REFDELAY are assumed to have transfer functions of 0 dB and 0 degree phase distortion. S_ϕ and R_ϕ are the effects of instrumentation in each channel.

The transfer function of the signal source is S-R, so regrouping, and solving for (S-R) one gets:

$$(S-R) = K_a(S_{meas} - R_{meas}) - SIGGAIN - SPROBE - 0 + REFGAIN + RPROBE + 0 - (S_\phi - R_\phi) \quad (3)$$

the 0s in the above equation are the effects of SIGDELAY and REFDELAY. The quantity $(S_\phi - R_\phi)$ is the transfer function of the effects of instrumentation on the system exclusive of sensors or external gain added. This quantity is called SCAL, the system calibration transfer function. Reducing the above equation becomes:

$$(S - R) = K_a\ (S_{meas} - R_{meas}) + REFGAIN - SIGGAIN + RPROBE - SPROBE - SCAL \quad (4)$$

The network analyzer can scale its output so that its 80 dB dynamic range brackets the data. Changes in this value changes $(S_{meas} - R_{meas})$ by a value equal in magnitude but of opposite sign. Therefore, adding this value, one gets the original equation:

$$(S - R) = K_a(S_{meas} - R_{meas}) + REFGAIN - SIGGAIN + RPROBE - SPROBE + NADR - SCAL \quad (5)$$

A similar derivation applies to the phase. Note that the phase of any gains in either channel are 0. The equation used for phase correction is:

$$\text{Corrected Phase} = K_\phi(PS_{meas} - DR_{meas}) + PHRPROBE - PHSPROBE - PHCAL \quad (6)$$

where

K_ϕ = conversion factor from millivolts to degrees for the network analyzer (here $K_\phi = .1$)

PS_{meas}-PR_{meas} = Phase in millivolts of signals sensed at the reference and signal channels. This value is the raw phase data received by the corrections task.

PHRPROBE = reference probe phase

PHSPROBE = signal probe phase

PHCAL = phase of instrumentation exclusive of probes, amplifiers or attenuators. This is the phase of the system.

Again, consider Figure 2-1. The phase output of the network analyzer is:

$$K_\phi(PS_{meas} - PR_{meas}) = \phi_S + PHSPROBE + PHSGAIN + PHSDEL + \phi_{S\phi} - \phi_R - PHRPROBE - PHRGAIN - PHRDEL + \phi_{R\phi} \quad (7)$$

where

PHSGAIN = phase of the signal gain = 0 as noted above

PHRGAIN = phase of reference channel gain = 0 as noted above

PHSDEL, PHRDEL = Phase of reference and signal delays. These are both zero due to the fact that on this system delays will be introduced by extra cabling, which, in and of itself, has phase = 0.

ϕ_S, ϕ_R = phase of the signal source at the signal and reference channels, respectively.

Since $(\phi_S - \phi_R)$ is what we want, by substituting and regrouping one can get

$$\phi = (\phi_S - \phi_R) = K_\phi(PS_{meas} - PR_{meas}) - PHSPROBE - 0 - 0 + PHRPROBE + 0 + 0 - (\phi_{S\phi} - \phi_{R\phi}) \quad (8)$$

The quantity $(\phi_{S\phi} - \phi_{R\phi})$ is the phase of the system exclusive of probes, gains, delays, etc., it is, in fact, the phase of the system cal, otherwise known as PHCAL. So reducing, one gets

$$(\phi_S - \phi_R) - K_\phi(PS_{meas} - PR_{meas}) - PHSPROBE + PHRPROBE - PHCAL \quad (9)$$

To do a system cal, the reference and signal sensors are removed from the configuration, i.e., RPROBE, SPROBE, PHRPROBE, and PHSPROBE are removed from the respective equations. Therefore, solving these equations for SCAL and PHCAL, one gets:

$$SCAL = K_a(S_{meas} - R_{meas}) - REFGAIN - SIGGAIN + NADR \quad (10)$$

and

$$PHCAL = K_\phi(PS_{meas} - PR_{meas}) \quad (11)$$

To calibrate a probe, the reference probe is removed from the system - that is the signal is run thru the system as shown in Figure 2-2.

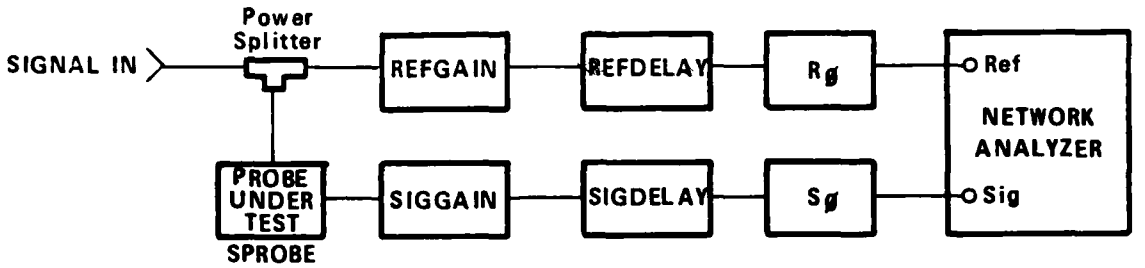


Figure 2-2. Probe Calibration Configuration

Solving for SPROBE and going through the aforementioned derivations, one gets:

$$SPROBE = K_a(S_{meas} - R_{meas}) + REFGAIN - SGAIN + NADR - SCAL \quad (12)$$

and

$$PHSPROBE = K_\phi(PS_{meas} - PR_{meas}) - PHCAL \quad (13)$$

When a multicycle test sequence is run, a signal-to-noise ratio calculation is made. This calculation represents the ratio of the ambient noise at the test point to the signal plus the ambient noise at the test point. The ambient noise measurement is the ratio of the signal at the test point to a constant reference level, provided by the reference synthesizer in the CWII system. This can be abbreviated to

$$N_a = \frac{S_N}{K}, \text{ or } S_N - K \text{ in dB}$$

and the signal-plus-ambient noise at the test point, which is the ratio of the signal at the test point to the signal received at the reference sensor, can be abbreviated

$$S_a = \frac{S_t}{S_{REF}}, \text{ or } S_t - S_{REF} \text{ in dB}$$

This is the test measurement. What is wanted is the ratio of S_t to S_N,

$$S_{NR} = \frac{S_t}{S_N}, \text{ or } S_t - S_N \text{ in dB}$$

When an ambient noise measurement is taken, K is the value for reference gain added in the primary menu. In order to calculate S_{REF}, a special measurement known as a reference sensor calibration is made. This is done whenever the test configuration is changed, or on some similarly frequent schedule. This measurement is the ratio of the signal detected at the reference sensor to a constant (K_2) reference level, again provided by the reference synthesizer. This can be abbreviated

$$S_b = \frac{S_{REF}}{K_2}, \text{ or } S_{REF} - K_2 \text{ in dB}$$

K_2 is the reference gain added from the primary menu when this measurement is taken, and is saved for the signal-to-noise calibration.

Now, we have (S_N-K), (S_t-S_{REF}) and $(S_{REF}-K_2)$, or N_a, S_a, and S_b, and we have saved the constant values K and K_2.

$$\begin{aligned} S_a + S_b - N_a &= (S_t - S_{REF}) + (S_{REF} - K_2) - (S_N - K) \\ &= S_t - S_N - K_2 + K \\ S_t - S_N &= S_a + S_b - N_a + K_2 - K \end{aligned}$$

The reference sensor calibration is saved on SY: in UIC[200,1]. The filename is RFSNS.CAL. The ambient noise and test data are extracted from the first and second parts of the corrected data file (CD:[200,1] CORECT.DAT). The reference sensor calibration reference gain constant is saved in record 23 of the primary menu file, and the ambient noise reference gain constant is extracted from record nine of the primary menu file.

2-3 CRT DISPLAY TASK ('CRT')

CRT provides the interface between the operator and the system. The program is run by the initialization indirect command file, but is inactive until it receives a data block from CORECT.

Initially, a data block is received which contains a description of the front panel settings of the measurement system (a panel data block). The information consists of the test type (TFA, TFC, etc.), the decade switch settings and the cycle number. The test type indicates to the CRT task whether phase data were taken. The decade switch settings are used to determine which decades contain measurement data so the CRT graph can be drawn accordingly, and how many points will be in each decade. These are decoded as A = 0, B = 25, C = 50, D = 100, E = 250, F = 500, and G = 1000.

Next, the operator is prompted for the types of plots to be displayed. Ambient noise, pick-up noise, amplitude and phase plots can be overlaid. At this time, CRT requests the IEEE-488 control task and suspends itself. The IEEE control task, ODL, checks for the presence of an IEC-11A bus controller module. If the IEC-11A status is good, ODL proceeds to validate the contents of the IEEE-488 device control menu. After the menu entries are validated, ODL issues bus commands to set the selected devices, verifies the settings, resumes CRT and suspends itself.

The primary and secondary menu entries are then verified and changed if necessary. The primary menu entries are displayed first. If approved, the program continues with the secondary menu. Otherwise, the operator is prompted for new entries to the menu.

The secondary menu entries are displayed. Again, the old entries are displayed and if not approved the operator is prompted for new entries.

When the menu processing is completed, the CRT graphics parameters are set to their default values. This is accomplished with the escape sequence '<ESC>*mR'. These values are shown in Table 2-1.

Next, the axes are drawn according to which plots were requested for the frequency scale, which decades contain data. The screen unit coordinates for the corners of the graph are always (49,119), (669,119), (669,335), and (49, 335). The phase

Table 2-1. Default HP-2648A graphics parameters

Parameter	Default Value
Pen Condition	up
Line Type	1 (solid)
Drawing Mode	set
Relocatable Origin	0,0
Text Size	1
Text Direction	1
Text Origin	1 (left, bottom justified)
Text Slant	0 (off)
Graphics Text	off
Graphics Video	on
Alphanumeric Video	on
Graphics Cursor	off
Alphanumeric Cursor	on
Rubber Band Line	off
Zoom	off
Zoom Size	1
Autoplot	off
Autoplot Menu	clear
Compatibility Mode	
Page Full Straps	0 (out)
GIN Strap	0 (CR only)

labels appear on the right y axis and are always the same. The amplitude axis (the left y axis) is labeled according to the value of the mid-range line. This is calculated by the following equation:

$$MID\ (dB) = -GS + GR + NADR - RV_s + RV_r \qquad (14)$$

where

GS = Signal Gain added

GR = Reference Gain added

NADR = Network Analyzer Display Reference

RV_s = Representative Value of Signal Probe Calibration

RV_r = Representative Value of Reference Probe Calibration

The values GS, GR and NADR are taken from the menu entries made by the operator. The values RV_s and RV_r are taken from the probe calibration files specified by the Signal Probe and Reference Probe menu entries. These files are opened and a representative amplitude value extracted from each. This is done to insure that the effect of applying corrections from these files does not force the plot off scale. The computed mid-line value is then saved in the primary menu file. The amplitude scale for the plot is +40 dB from the mid-line value.

When the axes have been drawn and labeled, a legend is written below the graph which describes the test being run.

A global flag is set to indicate that CRT is now ready to accept data from CORECT. The points to be plotted are received by CRT one at a time in twelve byte records each containing a frequency, an amplitude and a phase value.

Frequency values are converted to screen units by the following equation:

$$\text{Frequency} - \left(\frac{(\text{Log}_{10}\ (\text{last frequency point}) + \text{Log}_{10}\ 10^{-6})}{\text{Log}_{10}\ (\text{maximum frequency value})} - \frac{\text{Log}_{10}\ (\text{minimum frequency value})}{\text{Log}_{10}\ (\text{minimum frequency value})} \right) \times (669 - 49) + 49 + .5 \quad (15)$$

Phase values are converted to screen units, if a phase plot was requested, using the following equation:

$$\text{PHS} = \left[\left(\frac{(\text{Last phase value}) - (-180.)}{90.} \right) \left(\frac{(335 - 119)}{\frac{(180. - 180.)}{90.}} \right) + .5 \right] + 119 \quad (16)$$

Amplitude values are converted to screen units using the following equation:

$$\text{Amp} = \left\{\left[\frac{(\text{Last amplitude value}) - (\text{minimum dB value})}{10.}\right]\left[\frac{335 - 119}{8.}\right] + .5\right\} + 119. \quad (17)$$

As each phase point is plotted, it is saved as the last point and a line is drawn from it to the new point after it has been converted to screen units.

Before an amplitude point is plotted it is checked to see if the screen value falls outside of the axes lines (top or bottom). If so, a calculation is made to find the point of intersection, with the axis, between the point in bounds and the point out of bounds. The line is drawn between the point in bounds and the point of intersection. This is also the case in drawing from a point out of bounds to a point in bounds. Figure 2-3 illustrates this feature.

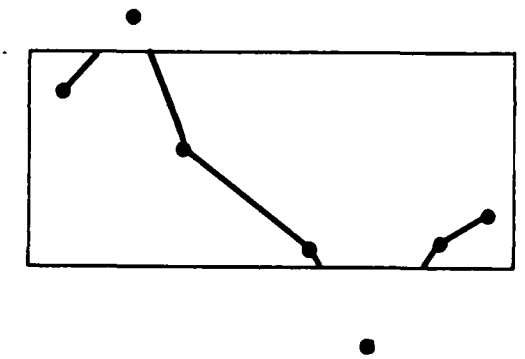


Figure 2-3. Example of Out-of-Range Point Interpolation

Lines between all points are solid. To differentiate between the lines, a symbol unique to the type of plot will be printed at evenly spaced intervals. The symbols used will be:

A - Ambient noise line

P - Pickup noise line

M - Amplitude line

F - Phase line

Should a test be aborted for any reason, a message will be displayed on the CRT screen indicating to the operator the

reason for aborting. The reason will be displayed as a code number which is described in Appendix A of the Operating Manual.

2-4 INVERSE TASK ('INVERS')

INVERS generates an inverse Fourier transform of the frequency domain data acquired by the measurement subsystem. The method used is a variation of the Guillemin Impulse Train method and will be discussed in greater detail later. The inverse transform also folds in a Threat waveform and a ninth order Butterworth filter. The resulting time domain waveform contains all three above-mentioned components.

INVERS is event-driven and executes asynchronously with the other data acquisition tasks. The task is compute bound, and runs at a low priority (49). The task is activated by CORECT which sends INVERS a message via a send message directive. This message contains either a panel data block (PDB) or an end-of-data message. The PDB message indicates the start of an inverse transform, whereas an end-of-data message indicates that no transform is to be calculated. To generate an inverse transform, the task must retrieve from the primary menu file the name of the Threat waveform file selected by the operator, the maximum time domain value and the B-field to E-field conversion factor. The points in the time domain to which the inverse is generated are then selected, the selected threat waveform file is opened and CORECT is signalled that INVERS is ready for data. Data are transferred from CORECT to INVERS via RSX-11M Send/Receive directives using global flags to signal when INVERS is ready and when CORECT has sent data. This allows asynchronous operation of both tasks.

When the inverse transform is completed, the Parseval energies,

$$\int_{-\infty}^{\infty} |f(t)|^2 dt \quad \text{and} \quad 2\int_{0}^{\infty} |F(\omega)|^2 d\omega$$

are calculated for the time and frequency curves, respectively. Since the square of a straight line is a hyperbola, a hyperbolic

rather than trapezoidal integration method is used. The results are compared using the standard error formula

$$\frac{X - Y}{X + Y}$$

and Parseval values and the ratio are stored in the primary menu. Then the file, CD:[200,1]INVERS.DAT;1, is created and the time domain data are written to it as eight-byte records containing two floating point numbers. Each record consists of a time value in seconds and an amplitude point. There are always 512 records in INVERS.DAT.

When INVERS finishes, it waits for two global event flags. One indicates that CORECT is finished and the other indicates that CRT is done. When both of these flags are set, all front-end processing is complete and the data are ready for post processing. The following operations are performed at this time:

- The operator is asked if a hard copy plot is desired and to enter a comment text.
- ODL is resumed and turns off all the ODL-5Bs. Then it terminates, releasing the IEEE-488 bus. When the bus is released, all the other devices on the bus go to front panel or standby operation.
- An entry is made in the mission file.
- The operator is prompted to determine which files are to be saved on tape and disk. The hard copy plots and tape files are spooled and the files to be saved for analyst are written to the analyst UFD [200,1] on the classified data disk.
- If a CRT plot of the inverse transform has been requested, the auto-plot task is started.

NHCPLT spools plots. The task which spools tape files is called STRTTP, and the one which saves files for analyst use is called ANLSAV. These tasks are installed when the data acquisition system is initialized (by the UP indirect command file) and are activated by CORECT when it receives the panel data block from the measurement system. The tasks receive messages from INVERS via system message directives. The

messages consist of a data type indicator and a filename. The value of the data type indicator tells the tasks what kind of data they are dealing with, and the filename becomes the output filename. The data type indicator has the following values and meanings:

0 - the ambient noise portion of a multicycle test.
1 - the test data portion of a multicycle test
2 - the pickup noise portion of a multicycle test
3 - inverse transform data
4 - transfer function calibration data, response function calibration data or probe calibration data
5 - reference sensor calibration data
6 - signal-to-noise ratio data
7 - end of test
8 - test data from a single-cycle test

If the data type indicator is 0, 1, 2, 3, 6 or 8, the message filename is as follows: the first byte is the test facility code from the secondary menu, the next four bytes are the test sequence number from the primary menu, and the last three bytes are the Julian date. If the indicator is 4 or 5, the message filename is as follows: if the test is a reference sensor calibrator's measurement, the first five characters are "INREF". If the test is a system response function calibration, the first five characters are "SYSRF". If the test is a system transfer function calibration, the first five characters are "SYSTF". If the test is a probe calibration, the first five characters are the first five characters of the probe calibration file ID. In all cases, the last three characters are the Julian date.

The method used for computing inverse transforms is a variation of Guillemin's Impulse Train Method[1]. Numerically its results are equivalent to a Fourier integral transform of

---

[1]Ernst A. Guillemin, Theory of Linear Systems, New York: John Wiley and Sons, Inc., 1963, pp. 387-393, pp. 509-510.

sampled data, but it has the speed advantage of operating on a relatively small number of terms. The justification for the Impulse Train Method for the inverse transform follows.

The equation for the inverse Fourier transform is

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(i\omega)\ e^{i\omega t}\ d\omega \tag{18}$$

where $F(i\omega)$ is complex with real and imaginary parts designated $R(\omega)$ and $X(\omega)$ respectively. Consider initially only the imaginary part. Let $X(\omega)$ be the function of angular frequency with n samples of $X(\omega)$ at ωj; $X(\omega j)$, $j = 1, 2, \ldots n$. $X(\omega)$ is approximated by straight line segments between each sample; call this $g(\omega)$. Let $h_j(\omega)$ be the jth line segment of $g(\omega)$:

$$h_j(\omega) = \begin{cases} a_j + b_j\omega, & \omega_j \le \omega < \omega_{j+1} \\ 0 \quad \text{elsewhere} \end{cases} \tag{19}$$

$$j = 1, 2, \ldots, n-1$$

Then

$$\sum_{j=1}^{n-1} h_j(\omega) = g(\omega) \tag{20}$$

By using the Guillemin algorithm, only a simple summation is needed in order to evaluate the contribution from the *imaginary part of* $F(i\omega)$. *It is not necessary to actually* integrate the approximating function. The Guillemin technique can be used with polynomials of order n, assuming only that $F^k(i\omega)\ 0 \rightarrow$ for $k=0, 1, \ldots n$ as $\omega \rightarrow \pm\infty$. In this case linear interpolation is used, and $X(\omega)$ and $X(\omega)$ are assumed to go to 0 as ω goes to $-\infty$ and $+\infty$. The algorithm is developed in the following way, integrating by parts twice.

$$\int_{-\infty}^{\infty} X(\omega)\ e^{i\omega t}\ d\omega = \frac{X(\omega)}{it} e^{i\omega t}\Big|_{-\infty}^{\infty} - \int_{-\infty}^{\infty} \frac{X'(\omega)\ e^{i\omega t}}{it}\ d\omega$$

$$= -\frac{X'(\omega)}{(it)^2} e^{i\omega t}\Big|_{-\infty}^{\infty} + \frac{1}{(it)^2} \int_{-\infty}^{\infty} X''(\omega)\ e^{i\omega t}\ d\omega$$

Now approximate $X(\omega)$ by $g(\omega)$, which is a broken line function as for the linear transform (see Figure 2-4).

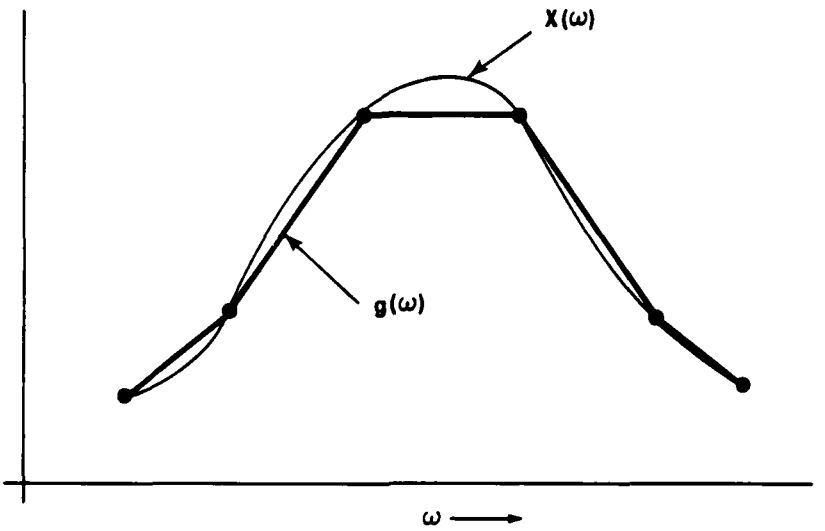


Figure 2-4. The Approximating Function $g(\omega)$

Let $h_j(\omega)$ be the jth line segment of $g(\omega)$:

$$h_j(\omega) = \begin{cases} a_j + b_j\omega, & \omega_j \leq \omega < \omega_{j+1} \\ 0 \text{ elsewhere} \end{cases} \tag{21}$$

$$j = 1, 2, \ldots, n-1$$

Then

$$g(\omega) = \sum_{j=1}^{n-1} h_j(\omega) \tag{22}$$

Now

$$g'(\omega) = \sum_{j=1}^{n-1} h'j(\omega) \tag{23}$$

$$h'(\omega) = \begin{cases} b_j, & \omega_j \leq \omega < \omega_{j+1} \\ 0 \text{ elsewhere} \end{cases} \tag{24}$$

and g' is a step function (see Figure 2-5).

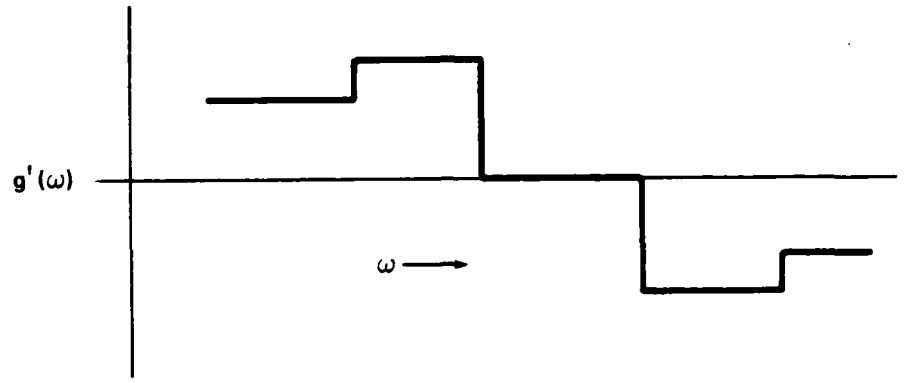


Figure 2-5. Derivative of g(ω)

Now

$$g''(\omega) \quad \sum_{j=1}^{n-1} \quad h''_j(\omega) \tag{25}$$

and if $\delta(\omega)$ is the Dirac delta function, then

$$h''_j(\omega) = \begin{cases} b_j \delta(\omega-\omega_j) \text{ at } \omega_j \\ -b_h \delta(\omega-\omega_{j+1}) \text{ at } \omega_{j+1} \\ 0 \text{ elsewhere} \end{cases} \tag{26}$$

$$j = 1, 2, \ldots, n-1$$

by virtue of the fact that $\int h''_j(\omega)\, d\omega = h'_j(\omega)$.
Therefore

$$g''(\omega) = b_1 \delta(\omega-\omega_1) + (b_2 - b_1)\ \delta(\omega-\omega_2) \tag{27}$$

$$+ (b_3 - b_2)\ \delta(\omega-\omega_3) + \ldots + (-b_n)\ \delta(\omega-\omega_n).$$

So

$$\int_{-\infty}^{\infty} X(\omega)\ e^{i\omega t}\ d\omega = \frac{1}{(it)^2} \int_{-\infty}^{\infty} X''(\omega)\ e^{i\omega t}\ d\omega \tag{28}$$

$$\cong \frac{1}{(it)^2} \int_{-\infty}^{\infty} g''(\omega)\ e^{i\omega t}\ d\omega \tag{29}$$

$$= \frac{1}{(it)^2} \left(b_1\ e^{i\omega_1 t} + \left(b_2 - b_1\right) e^{i\omega_2 t} + \ldots + b_n\ e^{i\omega_n t} \right) \tag{30}$$

Similarly,

$$\int_{-\infty}^{\infty} R(\omega)\ e^{i\omega t} \cong \frac{1}{(it)^2} \left(c_1\ e^{i\omega_1 t} + \left(c_2 - c_1\right) e^{i\omega_2 t} + \ldots + c_n\ e^{i\omega_n t} \right) \tag{31}$$

where c_j is the jth second derivative (impulse value) of $R(\omega)$.

Since the limits of integration are from $-\infty$ to $+\infty$ and $R(\omega)$ and $\cos\ \omega t$ are even functions while $\sin\ \omega t$ is an odd function, the total contribution from the real part is

$$-\frac{2}{t^2}\left[c_1 \cos\ \omega_1 t + (c_2 - c_1) \cos\ \omega_2 t + \ldots + c_n \cos\ \omega_n t\right] \tag{32}$$

Likewise, since $iX(\omega)$ is an odd function, the total contribution from the imaginary part is

$$+\frac{2}{t^2}\left[b_1 \sin\ \omega_1 t + (b_2 - b_1) \sin\ \omega_2 t + \ldots + b_n \sin\ \omega_n t\right] \tag{33}$$

Scaling the integrals by $\frac{1}{2\pi}$ then yields the inverse transform in impulse form as

$$f(t) \cong \frac{1}{\pi t^2}\left[b_1 \sin \omega_1 t + (b_2 - b_1) \sin \omega_2 t + \ldots + b_n \sin \omega_n t - c_1 \cos \omega_1 t - (c_2 - c_1) \cos \omega_2 t - \ldots - c_n \cos \omega_n t\right] \quad (34)$$

If $f(t) = 0$ for $t < 0$, the function is considered causal. Physical systems of the cw type may be so considered, and it can be shown[2] that for a causal system

$$f(t) = \frac{2}{\pi}\int_0^\infty R(\omega) \cos \omega t \, d\omega = -\frac{2}{\pi}\int_0^\infty X(\omega) \sin \omega t \, d\omega \quad t<0 \quad (35)$$

Since

$$\int_{-\infty}^{\infty} X(\omega)\, e^{i\omega t}\, d\omega = 2\int_0^\infty X(\omega) \sin \omega t \, d\omega \quad (36)$$

$$= -\frac{2}{t^2}\left[b_1 \sin \omega_1 t + (b_2 - b_1) \sin \omega_2 t + \ldots + b_n \sin \omega_n t\right]$$

then

$$f(t) = -\frac{1}{\pi}\left[2\int_0^\infty X(\omega) \sin \omega t \, d\omega\right] \quad (37)$$

$$= +\frac{2}{\pi t^2}\left[b_1 \sin 1_1 t + (b_2 - b_1) \sin 1_2 t + \ldots + b_n \sin 1_n t\right] \quad (38)$$

---

[2]Anthanasios Papoulis, The Fourier Integral and Its Applications, (New York: McGraw Hill Book Co., Inc., 1962), pg. 13.

Operation. As indicated earlier, this task uses a subroutine which is a variation of Guillemin's Impulse Train Method. The subroutine uses only the imaginary portion of the frequency function; hence it is a sine inverse. It is called to operate on all time elements (inner loop) for each frequency data point (outer loop). Thus, with each entry it augments a partial sum for each time point allowing it to operate in near real time. This results in the completed inverse being available as soon as possible after corrections have been applied to the final data point. The amplitude of time 0 is forced to 0 since the cw system does not record dc voltage.

Upon the first entry, a trigonometric sine function table is created with increments of one degree. For any given sine argument with times greater than or equal to 100 ns, a linear interpolation is performed between bracketing values of the table. For time less than 100 ns the FORTRAN library SIN function is used directly because any inaccuracy resulting from the table look-up operation is amplified by the scaling factor $1/\pi t^2$. This inaccuracy is tolerable above 100 ns where the table look-up feature saves considerable time.

As stated before, to use this method it is assumed that $X(\omega)$ and $X'(\omega)$ go to 0 as ω goes to $-\infty$ and $+\infty$. In order to make this true the integral is calculated as if there were a point (0,0) before the first frequency point and a point $(10*\omega_n,0)$ after the last frequency point at ω_n.

2-5 HARD COPY PLOTTING TASKS ('NHCPLT' AND 'NPSPLR')

NHCPLT and NPSPLR interface between the data collection tasks and the hard-copy plotter. NHCPLT formats the data and builds the data header. NPSPLR spools the header information and data points to the plotter.

NHCPLT receives the Panel Data Block from CORECT, then waits until INVERS has finished processing the data and has sent the filename to be plotted. NHCPLT then builds the plot file. The main and secondary menus are opened and the data file header is built. See paragraph 4-2.2.2 for the data header

description. The data points are then copied from the input disk file to the plot file. NHCPLT then sends the filename to NPSPLR using the system send directive.

When NPSPLR receives the plot filename and data type from NHCPLT, it initializes the plotter and waits for the operator to prepare the plotter. When the plotter is ready for operation, it writes the data header on the top portion of the plotting paper. Then it plots the desired data domains. Frequency-Magnitude, Frequency-Phase and Time-Amplitude plots may be generated. NPSPLR recycles through its procedures until no more plot filenames have been queued by NHCPLT.

2-6 TAPE OUTPUT TASKS ('STRTTP' AND 'TPWRTR')

STRTTP and TPWRTR write data files to cassette for archival storage. See paragraph 4-5.1 for a description of the cassette file structure. STRTTP sets up the file to be spooled to tape by formatting the file to make it compatible to the data files used in the Interactive Analyst Package. It receives the Panel Data Block from CORECT, then waits until INVERS sends a data filename and data type. Only the files that the operator has requested to be placed on tape are processed. STRTTP then opens the menu files to extract the needed header information. (Paragraph 4-2.2.2 contains the data file header description.) The header information and data are copied to a new data file, then the filename and data type are sent to TPWRTR via the send directive.

TPWRTR initializes the tape unit, dequeues a message from STRTTP, opens the output file, then determines the number of records in the file. The tape identification number is compared to the tape number in the data header; if they don't match the operator is informed of the error. The room remaining on the tape is compared to the number of records required by the data file. If there is not enough empty tape, the operator is requested to replace the tape with an initialized tape. The tape number in the data header is increased by one to match the new tape number. TPWRTR then copies the data file to the

cassette. While the data is being transferred, the minimum and maximum values are calculated, then placed into the file header on the tape. TPWRTR will continue processing the data files until no more filenames are in the receive directive queue.

2-7 DISK SAVE TASK ('ANLSAV')

ANLSAV writes the requested data files to disk for use in the Interactive Analyst Package. It receives the filename and data type which INVERS has sent via the send directive. ANLSAV opens the main and secondary menu files to obtain the information needed to form the data file header (Paragraph 4-2.2.2 describes the data file format.) It then copies the data file header and data to UFD [200,2] on device CD:. ANLSAV continues copying the data files until there are no filenames in the receive directive queue.

2-8 SET UP THE DATA LINKS TASK ('ODL')

ODL controls the Optical Data Links (ODL-5B) and VHF switch. It is requested to run by CRT after the type of plots have been selected. CRT pauses until ODL has completed setting up the instrumentation. ODL opens the ODL menu file and requests the operator to update the entries. (See paragraph 5-3.2 of the CW Measurement System Operating Manual for a description of the ODL menu entries.) First, ODL will set switch A and B on the VHF switch to the desired channel. It then turns on the requested data ODL-5B and sets it up. A five second pause occurs after the ODL-5B is turned on before it will accept any further commands. The reference ODL-5B is then turned on and set up. If no errors have occurred, ODL restarts CRT and pauses. When data collection is completed or the test is aborted, INVERS restarts ODL which turns off the ODL-5B units.

2-9 MISSION FILENAME TASK ('MISNAM')

MISNAM is run by the startup command file whenever the system is booted. It asks the operator to enter a mission filename. The extension .MIS is appended to the name entered by

the operator. If the mission filename is accepted, the secondary menu is opened and the name is entered into the tenth record. INVERS will try to open the mission file specified in the menu after all data processing is completed. If the file doesn't exist, a new file is created. Every time a data set is successfully processed, the filename along with key fields in the menus are written to the mission file. The Interactive Analyst Package routine, MIS, will list the contents of the mission file to the operator's console.

2-10 PROM PROGRAMMING TASKS ('DEL' AND 'AMPL')

After accepting the data that will be programmed onto the PROM, and checking with the operator for its validity, the data is formatted and written to the PROM programmer.

To start programming the PROM, the letter 'X' must be sent to the programmer first so that the automatic speed selection function of the PROM programmer can determine the input baud rate. A pause of approximately 2.5 seconds follows due to the time the PROM programmer takes to select this baud rate. Next, the address span to be programmed (i.e., 000-3FF) is sent to the programmer in ASCII. Each ASCII character sent to the programmer must be followed by a wait of approximately 1/30 second. Following the six-digit address span, the letter 'P' is sent, which sets the programmer to PROGRAM mode.

If the PROM is not blank, the programmer sends back and ASCII <NAK>; otherwise, if everything is ready, an ASCII <ACK> <STX> sequence is sent. Once this process is complete, the data may be sent to the programmer starting with address 0 and continuing to address X'3FF'. The data for each address is sent as two ASCII characters with a time delay of 1/30 second after each character. After all the data are sent, the remainder of the PROM is padded with zeros. The programmer then programs the PROM and, if successful, sends back an ASCII <ACK>. If the programming was not successful for any reason, the programmer sends a <NAK>. The program will report to the operator whether the PROM was successfully programmed.

The tasks perform input range checks and checks to insure that frequency data is in ascending order. Once the PROM writing sequence begins, the handshaking between the M900 and the PDP-11 is checked for validity as described above, and also timed-out. This is done by setting a timer while waiting for the read of the M900 response to complete, using the RSX-11M Mark Time directive and checking various System Local Flags. (Local Flags are identical to Global Flags except that their scope is limited to the task itself.) A response time-out causes an error condition to be noted within the task and the read to be halted. If either a time-out or handshaking failure occur, an appropriate error message is logged to the terminal, and the operator is given the opportunity to abort the task or to retry. This feature allows the operator to correct "soft" errors (e.g., PROM not blank) and retry the writing step without having to re-enter the data.

2-11 ANALYTIC PROBE CALIBRATION TASK ('APROBE')

2-11.1 Background Discussion

Transfer functions of certain probes are obtained analytically rather than empirically (e.g., the MGL B-dot probe). Analytic transfer functions of electronic integrators may be multiplied to yield a composite transfer function.

The probes handled analytically by APROBE are the derivative type:

$V(t) = kF$ where

$V(t)$ = output voltage as function of time

k = a constant determined by the probe

$\dot{F} = \frac{d\ (\text{Field Unit})}{dt}$

In the frequency domain this transforms to

$V(\omega) = i\omega kF(\omega)$ volt - seconds where

$i^2 = -1$

$\omega = 2\pi \times$ frequency radians/second

The transfer function of the RC integrator is

$$\frac{V_{out}(\omega)}{V(\omega)} = \frac{1}{1 + i\omega RC} \qquad (39)$$

Note that if no integrator is used, RC = 0, and the integrator transfer function becomes unity.

The transfer function of the composite is therefore

$$\frac{V_{out}(\omega)}{F(\omega)} = \frac{i\omega k}{1 + i\omega RC} \qquad (40)$$

having amplitude and phase characteristics similar to Figure 2-6.

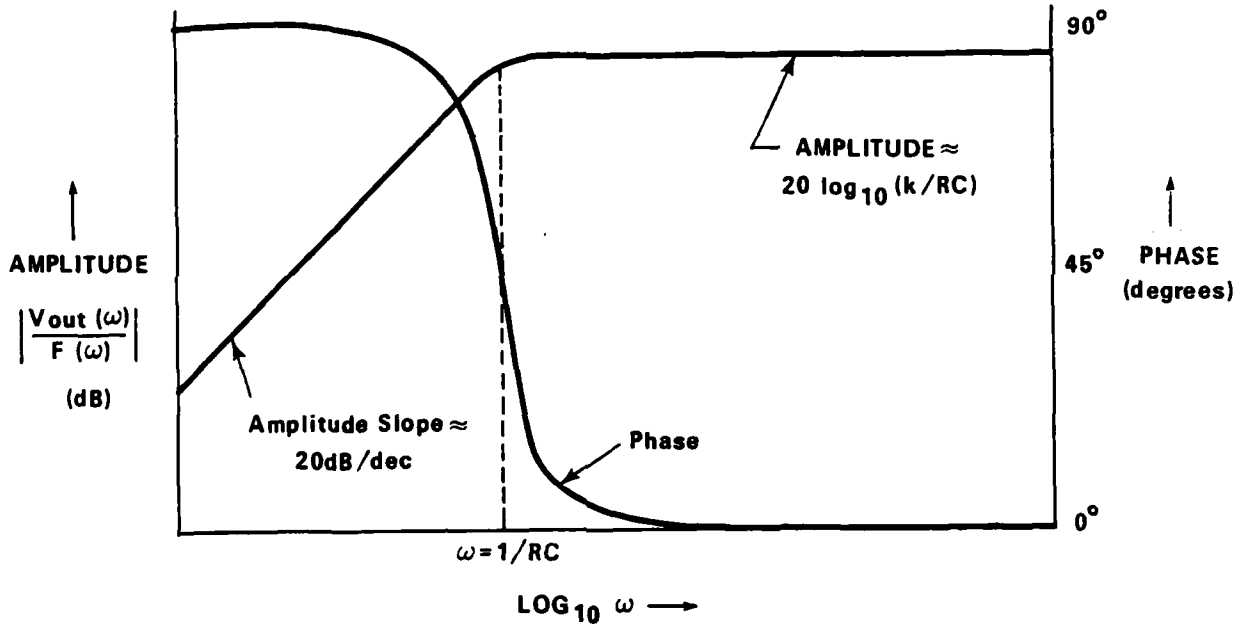


Figure 2-6. Transfer Function Amplitude and Phase of Analytic Probe with Integrator

2-11.2 Operation

The program opens the requested file as TYPE = 'OLD', implicitly assuming the file already exists. If the file exists, the operator is given the option to supersede with new

data or to exit. If he elects not to exit or if the file did not exist, a new file is opened and the operator is prompted for the scaling constant (k) and integrator time constant (RC). Having obtained this information, the program calls subroutine FGEN which generates 450 predefined frequency values. The program then generates the resultant transfer function as dB and degrees for each frequency and writes these values as real valued triplets to [200,1] PROBE ID.CAL;1 on volume SY: where PROBE ID is the probe identifier entered by the operator (e.g., B201).

After all such triplets have been written, a final triplet whose first value is -1 is written and the file is closed. This record, as in all data files generated by the system, indicates end-of-file to tasks which later read the file.

2-12 INPUT WAVEFORM AND BUTTERWORTH FILTER TASK ('THRTWV')

If a network's response to a unit impulse is convolved with an input waveform, the result is the waveform at the network output. Since convolution in the time domain is equivalent to multiplication in the frequency doman, and the frequency domain transfer function is the forward transform of a unit impulse response, one may inverse transform the product of an input waveform and transfer function in order to get the predicted output in the time domain.

Of particular interest is the input waveform of the type:

$$E_i(t) = A(e^{-\beta t} - e^{-\alpha t}) \text{ v/m} \quad 0<\beta<\alpha \tag{41}$$

where

$$A = 5 \times 10^4 \times \left(\frac{\alpha+\beta}{\alpha}\right)\left(\frac{\alpha}{\beta}\right)^{\left(\frac{\alpha}{\alpha+\beta}\right)} \text{v/m} \tag{42}$$

The analytic forward transform, $E_i(f)$, has the following amplitude and phase:

$$\left| E_i(f) \right| = \frac{A\ (\alpha - \beta)}{\left[(\alpha^2 + \omega^2)\ (\beta^2 + \omega^2) \right]^{\frac{1}{2}}} \quad \nu\text{-sec/m where } \omega = 2\pi f \qquad (43)$$

$$\phi_i(f) = -\tan^{-1} \left(\frac{\omega(\alpha + \beta)}{\alpha\beta\ \ \omega^2} \right) \quad \text{rad} \qquad (44)$$

On a log-log plot, the amplitude appears as Figure 2-7.

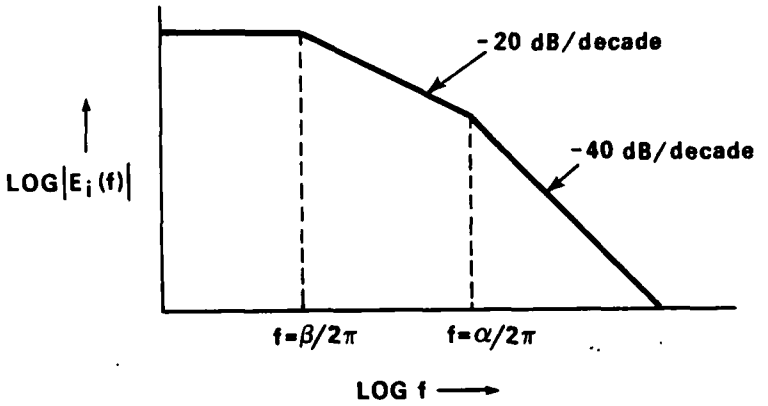


Figure 2-7. Amplitude of Transformed Input Wave

Prior to performing an inverse transform, the product is multiplied by a ninth order Butterworth filter in order to reduce truncation effects. The Butterworth amplitude is essentially unity except near the lower and upper cutoff frequencies. Beyond these frequencies the amplitude slopes at -54 dB/octave (nearly -180 dB/decade). It is convenient to apply the filter to the transform of the input waveform.

Finally, since the data is stored in dB and degrees, the product is converted to these units and stored on the classified data disk (CD:).

SECTION 3
INTERACTIVE ANALYST PACKAGE

3-1 INTRODUCTION

This section describes the tasks that comprise the Interactive Analyst Package (IAP). The processes provided by the IAP are:

Name	Process
ADD	Addition process. Performs an addition of two data sets in frequency or time domain.
AHD	Add header process. Add header to time and frequency domain files for CWII compatibility.
ANL	Analytical waveform process. Generates a time domain waveform based upon operator entries of coefficients of an analytical expression. This process issues subsequent prompts.
CTP	Cassette tape transfer process. Transfers data from cassette to the CD: disk for subsequent manipulation. Also transfers data from the CD: disk to the cassette.
CVT	Convert tape process. To convert tape files copied onto disk by TPC from the CWI file format to the CWII file format.
DED	Data Edit process. Performs editing on the data by allowing corrections, insertions, deletions and appendages to the data. DED will also accept entries in a tabular form for data in both frequency and time domain. Data can also be listed from an existing file using this process.
DIV	Division process. Performs a division of two data sets in frequency or time domain.

FTR Forward Fourier Transform process. Performs a forward transform using the Guillemin algorithm.

ITR Inverse Fourier Tranform process. Performs an inverse transform using the Guillemin algorithm.

LHD List header process. Lists all records of the header for a specified file.

LOF Log-off process. Logs the user off the system and allows the deletion or preservation of all or selected files created during the processing sequence. When no processing has been done LOF allows deletion of previously created data file.

LON Log-on process. Logs the user onto the system and allows the operation of the other IAP processes.

MIS Mission file listing process. To list the contents of a requested mission file on the screen.

MUL Multiplication process. Performs a multiplication on two data sets in frequency or time domain.

SCL Scaling Process. Performs scalar arithmetic on either time or frequency data files.

SUB Subtraction process. Performs a subtraction of two data sets in frequency or time domain.

TPC Tape copy process. Copy all of a cassette tape into a disk file. Primarily used for the conversion of old format tapes.

When the operator wants to use the analyst package, the following command should be entered. The proper tasks will be installed.

@ANAL <CR>

where <CR> is carriage return.

The commands used in the IAP are designed to give the analyst a flexible set of operations to manipulate data taken by the real-time data acquisition software. The commands used to run all of the processes are structured using a syntax very similar to that of the standard DEC RSX-11M system utilities and language processors, like PIP (Peripheral Exchange Program) and FORTRAN IV-PLUS.

The command syntax uses the concept of a process acting upon a source of input, yielding an output. The basic command syntax is as follows:

PRC output.list=input.list

where PRC is a three-character process name, input.list is a list containing specifier(s) of input source(s), and output.list is a list containing specifier(s) of the output(s) of the process. These lists can contain filenames, terminal or device specifiers, and process modifiers (called "switches") in various combinations, depending on what is valid for the process. Switches are indicators by which the process can be signalled of special conditions or of changes in the normal processing that the operator requires. Switches consist of a slash ("/") followed by a switch identifier and an optional list. The switch syntax is illustrated below:

/SID:parm.list

where SID is the switch identifier and parm.list is the list of parameters. Note that the parm.list is preceded by a colon and may or may not appear, depending upon whether parameters are required by the switch. If a switch requires more than one parameter in its parm.list, each parameter is separated by a colon. An example of a switch with a parm.list of six parameters is given below:

/SID:p1:p2:p3:p4:p5:p6

Switches can be applied to a command's input.list or output.list (or both), depending again on what is appropriate for a given process and switch.

Processes can use the information in the command line to complete the process, or can optionally prompt for more information, data, etc.

Processes generally work with disk or tape based data files. A file is identified by a fully qualified name of up to eight characters and an extension of up to three characters. Within the IAP, a convention provides a means of identifying the source of the data and the date of creation of the file.

A general rule of thumb should be followed in naming a data file and its extension. The name is an eight character name with a three character extension which is in the following form:

FSEQNJDT.EXT

where

F--- is a character A-Z or 0-9 representing a code to identify the test facility

SEQN is four integer numbers defining a unique test sequence number

JDT is three integer numbers defining the Julian date of the measurement. This value is appended to the facility code and sequence number by the software.

EXT is the three character extension where:

E - is one of the following six characters:

- A - indicating ambient noise
- U - indicating pickup noise
- C - indicating cw measured test data
- T - indicating time data
- P - indicating pulse measured test data
- F - indicating frequency data

X - is one of the following 3 characters:

C - indicating calculated data
D - indicating defined data
M - indicating measured data

T - is an "A" except in the case of converted CW1 data where a sequence number was used for more than one measurement. In this case, the last character will be incremented through the alphabet as many times as necessary.

The entire extension is also appended by the software. Calibration files saved for analyst use are an exception to the above convention. They will be named using the five-character sensor name found in the header. The Julian date and extension will be used as described above.

When an analyst is naming a file to be created by any operation, the first five characters can be used in any way as an identifier. These five characters must be unique from those used to name another file. The Julian date and an extension will be appended to the five characters as above.

The IAP software will store the data on disk to a device named CD:. This device is usually assigned to DL1:, and is used for data storage only. The software will assure that the data are stored on CD: so the operator need not specify this device name.

3-2 COMMAND STRING PROCESSOR

All of the IAP processes use a macro program to interpret their command lines. Each macro program is essentially the same except for the number of switches and input files that are allowed. The system routine CSI is called by the macro program. Each process command string is described in detail in Section 8 of the CW Measurement System Operating Manual.

3-3 ADD, MUL, DIV AND SUB

The basic arithmetic functions are essentially the same programs except they perform different operations. They allow the operator to apply a function of two data sets. The command line syntax for ADD, MUL, SUB and DIV is:

```
ADD
DIV
MUL   } outfile = infile1.ext,infile2.ext
SUB
```

Infile1 and infile2 are the data files which are to be multiplied, added, divided or subtracted.

Infile2 is optional.

Outfile is the resulting file.

In the case where only one input file is specified, the input data are simply copied to the output file. If a switch is specified for the output file, the output data are interpolated to the time interval or number of points per frequency decade specified by the switch; otherwise, time data are copied point-for-point to the output file. Frequency data are interpolated to the number of points per frequency decade specified in the file header.

In the case where two input files are specified, the output data file reflects the mathematical result of the applied function: addition, subtraction, multiplication or division. For time domain data, the output data are interpolated to a regularly-spaced interval, calculated either by dividing the minimum time spanned by the two files (the maximum of the respective minimum times to the minimum of the respective maximum times) by 500, or input by the operator as part of the command line. If the value input by the operator results in more than 1,000 intervals, an error message is generated and the number of intervals is set to 500.

For frequency domain data, the output data are interpolated to a number of frequency points per decade, either input by the operator or gotten from the header of the first input file. If either input file header indicates no frequency points for a decade, no output data is calculated for that decade.

Empty decades may occur between non-empty ones. The total number of frequencies may not exceed 1,000 in either the input or the output files.

The selected function is applied to the data, and the result is stored in the output file, along with an updated header (see paragraph 4-2.2.2 for header format). For addition and multiplication, the order of operation is important only insofar as default values such as points per frequency decade are taken from the first file specified in the command line. For subtraction, the second file specified is subtracted from the first, and for division the first file specified is divided by the second.

3-4 ADD HEADER PROCESS (AHD)

AHD adds a header to a CWI data file to make it compatible with the IAP. The command to run AHD is:

```
AHD outfile/switch=infile.ext
```

The switch specifies whether the outfile is to be a time or frequency domain file.

AHD opens both input and output files and interpolates the time or frequency values to the requested interval. It then calculates the maximum and minimum data values while copying the data to the new file. The following records are entered into the outfile data header:

Record	Field
1	Data domain
3	Date
4	Time
5	Input filename
7	Function code and date
12	Comments
13	Comments
23-28	Maximum and minimum data point values
37-42	Points per decade
45	Time domain Δt

The remainder of the header is left blank. All the data points are then copied to the output file.

3-5 ANL

ANL generates an analytical waveform in the time domain. ANL is run by entering the command string:

```
ANL outfile
```

The maximum time value and/or the desired number of points may be specified. To create the waveform, ANL will ask the operator to supply values for the following equation:

$$A\left[B\left(e^{Ct} + Dte^{Ct} + Ee^{Ft} \cos\left((2\pi Gt)-H\right) + J\right)\right]$$

After the constant coefficients have been entered, the time value and amplitude are calculated and written to the output file. The minimum and maximum time values are calculated. The appropriate values are written into the header records.

3-6 CTP

The cassette tape task allows analyst manipulation of data files between the disk and the cassette tapes. Using CTP, the operator can transfer files from disk to tape, from tape to disk or list the tape directory. The function to be performed when CTP is invoked is determined by the syntax of the command line. To transfer a data file from disk, enter the following:

```
CTP/TP:number[/NEW]=infile.ext
```

Infile.ext is the fully qualified name of the file to be transferred. This name must be unique on the tape. If the name is not unique, the operator is given three choices for eliminating the problem. One is to change the name of the file going to tape, another is to insert another tape, or, last of all, the operator may exit the program.

If the operator chooses to change the name of the file, a unique name of up to five characters must be entered. A three character representation of the current Julian date will be concatenated to the five character name entered. The extension remains the same as the original file specification.

When the operator desires to change the tape, the program reads the input from the terminal until the operator enters

a <CR>. Then the program tries again to write the r ıested file to tape.

To write a file from disk to tape, each 12 by· disk header record is copied to an 86 byte tape header reco . The data is converted from one triplet or one doublet (in ternal binary format) in a 12 byte disk record to two ASCII tri ets or three ASCII doublets in an 86 byte tape record. After e file is copied to the tape, the operator is given the option place the file name in the analyst log so that it can be delet from the disk directory at log off.

/TP:number in the above command line specifies e tape number to which the file is to be copied. This numb must agree with the number actually written on the tape. If the numbers are not the same, the operator is given the ch ce of changing the tape (as described earlier) or exiting the ogram.

The optional switch, /NEW, in the command line, specifies that the tape is to be initialized. When a ape is initialized, the requested tape number is written in th first six bytes of the first two tape records. The remainde of the two records are filled with blanks. These two records mprise the tape directory. If no file is specified in the comm d, the task will exit after the tape is initialized. Otherw e, the file is written to the initialized tape.

When a file is to be copied from tape to di , the following command line must be entered.

CTP[outfile]=infile.ext/TP: number

Outfile, when specified, is up to five characters used w h the three character Julian date to name the disk fil The extension will remain the same as that on the tape file. If no outfile is specified, the disk file is named the sam as the tape file requested. The filename must be unique on the isk or the task will exit. When the file transfer is comple , the filename is added to the analyst log.

Infile.ext is the fully qualified name of the le to be copied from the tape. The tape directory is sea hed to determine if the file exists on the tape.

/TP:number is a switch indicating the number of the tape from which to copy the file. If the number on the tape and the number specified are not the same, the operator is given the option of changing tapes or exiting the program.

To get a directory listing of the tape currently in the drive, the following command is used:

CTP [TI:/LP:] /DIR

[TI:/LP:] is an optional device specification requesting the terminal or line printer, if available.

The directory listing shows the tape number, the filenames in the directory, and the number of tape records occupied by each file. There is a maximum of eight files permitted in the tape directory.

Tapes are organized so the first two records are the directory of the files contained on that tape. Each directory record begins with the tape number comprised of a maximum of six digits. A maximum of 12 characters (eight characters, a period ('.') and a three character extension) can be used for a filename. File names are followed by an equal sign ('=') and an integer of up to four digits, indicating the number of tape records used by that file. There are no more than four files per directory record. The number of records used by each file are added together in CTP to determine if sufficient room is available to write the requested file to the tape. These numbers are also used to calculate the number of records to skip to get to the beginning record of a file to copy or to reach the next available record.

Each file on the tape begins with a record containing a '%0' and the filename. Then 57 records follow containing the file header. Only 12 of the available 86 bytes of tape record are used for each header record.

The data is in ASCII, 13 characters per data value with six data values per record. The file is ended with a '%/ END OF FILE' record.

Disk files are arranged with 57 ASCII records, each 12 bytes in length, containing the file header. Following the

header are data records of 12 bytes which contain one triplet or one doublet (depending on the data type) written in real internal binary format. The files are direct access so the end of file is determined when a read error occurs.

RSX's command string interpreter is used to check for proper syntax in the command line. The operator must have logged on using LON in order for CTP to run.

For communication to take place between the tape and the disk, the tape unit must be attached using a Wait Queue I/O command. When data are transferred from the tape to the disk, the proper function value must be sent to the tape unit to start sending. During the sending process, a five second timer is set so that when nothing is received from the tape, the send request is cancelled.

For data transfer from disk to tape, the function value to start the tape receiving must be sent to the tape. A status returned from the tape must be checked, until only the ready bit is set, before the program continues. A function value to stop the tape receiving must also be sent.

Files used by CTP are found on the disk assigned to device CD: in UFD[200,2].

3-7 CVT

The convert tape task is used to convert the CWI formatted tape files created by TPC into individual CWII formatted disk files.

Prior to invoking CVT, the operator must have logged on using LON. CVT can be started by entering CVT<CR>.

When CVT begins, the operator is prompted for the tape number to convert. The number entered is concatenated onto a string of zeros to create a six digit name. An extension of '.OTP' is added to make a fully qualified filename. The file to be converted is expected to be found on the disk assigned to device CD: in UFD[200,2].

CVT searches for a '%' in the first byte of each record. Whatever character is in the second byte determines the type of file to be created.

When the character is a '1', a 32 record "storybook" follows. These records are used to create a 57 record file containing the header to be used in the new CWII files. The test sequence number and the creation date are retrieved from the storybook. After converting the date to a three character Julian date, the operator is prompted for a one character test facility code. The test facilty code, the sequence number and Julian date are concatenated to form the filename which will be used for each file created for any data type using that sequence number. The files will be distinguished by their extension.

When the character following the % is a 2, the following data are ambient noise and the extension is '.AMA'. A 3 indicates test data and the extension is '.CMA'. '.UMA' is the extension used when the second character is a 4 specifying pickup noise. A 5 is used when inverse transform data are present. In this case, the extension is '.TCA'. Last of all, a 0 indicates the end of the current file.

Each filename is required to be unique. If this is not the case, the last character of the extension is incremented through the alphabet until a unique name is found.

The files consist of 57 12-byte ASCII records containing the header, followed by 12 byte records of real internal binary data values in triplets or doublets depending on the data type of the file. As each file is created, its name is placed into the analyst log. The entire tape file is converted in the following manner until no more data exists.

When a record beginning with '%' is found and the data are determined to be in the frequency domain, two triplets are assumed to be in each record. The data values are converted from ASCII to real internal binary and written to the new file in a 12 byte record with one triplet per record.

A group of data determined to be in the time domain is assumed to be in records of three doublets each. These values

are converted like the frequency data and written to the new file with one doublet per 12 byte record.

Minimum and maximum values of the data are calculated as the conversions are done. When the conversions for each data type are completed, the minimum and maximum values are written to the header.

3-8 DED

The data edit task has two primary functions. DED can be used to create a new file from tabular data entered by the operator or to edit an existing data file.

When creating a new file, DED is invoked by the following command line:

DED outfile/CR

Outfile is a five character name specified by the operator. The data entered by the operator is stored in a disk file using outfile, a three character representation of the current date and a three character extension concatenated together. The extension is 'TDA' for time domain data and 'FDA' for frequency domain data. After concatenation, the resulting filename must be unique.

Data editing is started by entering the following command line.

DED infile.ext

Infile.ext is a fully qualified filename consisting of up to eight characters in 'infile' followed by a three character extension. This name is used to identify the file to be edited.

Files created by DED, whether new or edited, are stored on the device assigned to CD: in UFD[200,2].

Once started, DED first determines if the operator has logged on using LON. Next, the command string interpreter, provided by DEC, is used to parse the input command line for valid syntax.

When creating a new file, DED ensures that the filename is unique by first opening the file as 'old'. If the open is

successful, the operator is informed the file already exists and the program exits.

If the open fails, the file is opened as 'new' and the operator is prompted to enter triplets (frequency, magnitude and phase) or doublets (time and amplitude) depending on the data type. The values can be entered in integer, real or E-formats. Each frequency or time value must be positive and must be greater than the previous frequency or time value entered (maintain monotonicity). When these conditions are not met, the operator is informed of the discrepancy and prompted to re-enter the value. As each data set (doublet or triplet) is entered, the values are compared with the existing minimum or maximum values. These existing values are updated if necessary.

When a <CTRL Z> is entered for the frequency or time value, the prompting is stopped. The header for the new file is created. If the file is in frequency domain, the operator is prompted to enter the number of points in each decade. These values are written to the header. Once the header is completed, the file is closed and the filename is entered into the analyst log.

For data editing, a menu of six processes is displayed. This menu is displayed again after the completion of each process except number six, EXIT.

The first process, LIST, reads each record of data, converts it to ASCII and prints it on the screen. This option is intended for use by the analyst to locate where data editing is necessary.

Process two is used to append data to the file. The operator must specify the location in the file where appending is to be done. The original data are copied to a temporary file up to the position of appending. Then as many data sets as desired can be entered, as long as they follow the restrictions as described for new data entry. When a group (one or more) of data sets has been appended, the operator is prompted for more appending. The original data are again copied to the temporary file to the next position for appending or the end of the file

if no more appending is requested. More than one appending process can be requested only if the selected positions occur sequentially in the file.

Data insertion can be accomplished be selecting process three. This process is identical to appending except the position selected by the operator comes after the inserted data, whereas the position comes before appended data.

Process four is selected when modification to a data set is desired. More than one data set can be changed if they are requested sequentially through the file. The original data are copied to a temporary file except for the data sets requested for modification. The operator is given the option to change each value in the data set. Entering a carriage return causes the value to remain unchanged.

The deletion process is started when five is selected. Selection of the first deletion position is required by the operator. With a single deletion request, any number of consecutive data sets can be deleted. Other data sets can also be deleted if requested sequentially. Except for the deleted data sets, the original data are copied to a temporary file.

Exiting from DED is accomplished using process six. First the program determines if a file was edited since DED was started. If it was, the operator is given the choice of deleting the original file. Then, if desired, the temporary file can be named the same as the original file. Otherwise, a five character name must be entered. A current three character Julian date and the proper three character extension are concatenated for a fully qualified name for the new file.

After the new filename has been determined, the temporary file is copied. As they are copied, the frequency or time values are checked to ensure they are monotonically increasing. If the check fails, a message is written to tell the operator, but it does not stop the file creation. Minimum and maximum values are also checked as the data are copied and then written to the header when the copy is completed. The original filename, 'DED', and the current date are written to

the header records five through seven. Points per decade are not updated.

When more than one process is requested during an edit session, the temporary file most recently created becomes the original file for the next process. After the last temporary file is copied to the new file, all the temporary files are deleted from [200,2] and the new filename is placed in the analyst log.

3-9 FTR

This task performs a forward Fourier transform on time domain data using the Guillemin's Pulse Train Method described in the section on the task INVERS. After the time data are read in, the program builds the frequency output array, evenly spacing the points within each decade and using the numbers of points per decade requested by the operator or the defaults. A warning is issued if the maximum frequency requested is greater than half the sampling rate of the time data (the Nyquist criteria) but processing continues.

Calculations of the integral proceed as if a point (0,0) exists before the first time point and a point (tn * 10,0) exists after the last point at time tn. Both real and imaginary values are calculated for each frequency point. After all points have been processed, the Parsevals are calculated and compared as described in the INVERS write up.

3-10 ITR

Using a frequency data file as input, this routine performs an inverse Fourier transform using the same method as the on-line task INVERS. After the frequency data has been read in, the time array is built using the number of points and maximum time requested by the operator or defaults of 500 points and a maximum time equal to the number of points minus one times one over the maximum frequency present.

The magnitude of time 0 is set to 0 and calculations proceed as if a point (0,0) existed before the first frequency

point (ωn * 10,0) existed after the last frequency point at frequency ωn. The frequencies are converted to radian frequencies and the program loops over all of the frequencies (inner loop) for each time point (outer loop).

When all of the time points have been processed, the Parseval energies and their ratio are calculated as described for INVERS and stored in the header of the output file.

ITR assumes that the data has no dc value, therefore it should not be used with data that has a dc value.

3-11 LHD

The list header task is used to list the contents of the header in a specified analyst file on the terminal screen. LHD is started with the following command line:

```
LHD infile.ext
```

Infile.ext is a fully qualified name with up to eight characters in the filename followed by a three character extension. The operator is required to have logged on using LON prior to invoking LHD. The file specified is expected to be on the disk assigned to device CD: in UIC [200,2]. LHD uses logical unit 1 assigned to the terminal to write the header. This unit number can be assigned to the line printer, if available, to get a hard copy of the file header.

The header titles and contents are printed in such a way that the entire header can be viewed on the screen at one time.

RSX's command string interpreter is used to parse the command line to be sure the syntax is correct.

Each of the header records read by LHD in the file are 12 byte ASCII records. The records are read into a 12 byte buffer and written with the appropriate title to logical unit 1.

3-12 LON AND LOF

LON and LOF log the operator onto and off the system allowing full use of the analyst package. LON sets the "log on" event flag and initiates the log task which records all the

analyst files created until the operator logs off. LOF sends a message to the logging task to initiate the shutdown of user activity and to allow the user to save or delete the files created during this session. Each IAP process checks the 'log on' event flag. The data filename log file is in UFD [2,2] on the system disk and is called LOGTSK.Q.

3-13 MIS

MIS lists the contents of the mission file. It is initiated by entering the command

MIS infile.mis

The contents of the mission file is information selected from the menus. It will be listed on the terminal in the following 80 column format, first line:

Columns	Field
1-4	Test number
6-13	Filename
15-18	Tape ID
20-28	Date
30-37	Time
39-42	Test type
44-55	Test point ID
66-73	Test engineer

Second line:

Columns	Field
1-80	Test comments

A blank line will separate each data set. The listing is divided into pages of 15 filenames per page. The operator must use <CTRL S> and <CTRL Q> to stop and start the output for closer examination.

3-14 TPC

The tape copy task is used to copy the contents of an entire cassette tape into a single disk file. The task is invoked by entering

TPC<CR>

TPC sends a function value to the tape unit so it begins to send a record from the cassette unit to the PDP-11/34. Each record is 86 bytes long containing only ASCII characters. When a record is sent, it is received in an 86 byte array which is then written to a sequential disk file with records 86 bytes in length.

As each record send is attempted, the status returned from the tape unit is checked. When the busy bit is turned off, the next send is begun. If a CRC error, receive error, or fault occurs, an error record is written to the disk file. The error record is '%/ TAPE RECORD ERROR DURING COPY'. When the tape leader is reached before the tape copy is complete, the error record '%/ END OF FILE NEEDED' is written. '%/' is used to begin the record for compatibility with the tape conversion routine. Prior to invoking the converson task (CVT), the DEC editor TECO must be used to correct the error records in the file.

When a request to send is issued to the tape unit, a five second timer is started so if no record is received within that amount of time the send request is cancelled and the task completes.

TPC begins by prompting the operator to enter the tape number provided LON has been used to log on. The number entered is concatenated onto a string of zeros to make a filename of six digits. An extension of '.OTP' is used on the filename. The file is written to the disk assigned to device CD: in UFD [200,2]. After the file is closed on the disk, the filename is entered into the analyst log.

SECTION 4
SYSTEM DATA STRUCTURES AND GLOBAL VARIABLES

4-1 INTRODUCTION

This section addresses the definition of data structures and global variables used in the CW Data Acquisition Subsystem software. The definition of data structures will start with a discussion of some operating system constructs used extensively throughout the real-time system. Discussion of disk and file structures used by the software will follow. Communication structures, specifically the message structures and RSX-11M Global flag definition follow the file structures. Global flags are used for intercommunication between tasks.

Other structures used for archival cassette storage and the PCU frequency table entries are discussed along with a discussion of 1-disk vs 2-disk system operation.

4-2 FILE STRUCTURES

The system, during data acquisition, may have as many as seven files open at a time. This section discusses some of the relevant system constructs, the file formats and access methods that are used.

4-2.1 Operating System Constructs

This information also appears in the RSX-11M Documentation Package. Reference to the relevant sections of the package is encouraged for a more detailed discussion of these and related constructs.

4-2.1.1 UFD's and UIC's. The term UFD is a mnemonic for User File Directory and UIC is a mnemonic for User Identification Code. Both constructs have identical format and are sometimes interchangeable. The form of a UIC or UFD is

[g,m]

where the brackets are required syntax and g and m are octal numbers in the range 1-377.

In order to be able to access portions of a disk in a random fashion (as opposed to sequentially scanning the disk looking for a particular file) a data structure called a directory is imposed on the disk. The directory contains various pieces of information concerning files including a file's name, its starting location, its block allocation, etc. By scanning the directory, a given file can be found much quicker than if the entire disk were to be scanned. In a multi-tasking environment, where many users may be accessing files on a disk simultaneously, it may be desirable to keep a user from accessing or writing upon another user's file(s). RSX-11M's file system uses the construct of multiple directories, one for each user or class of user, containing that user's files exclusively. Each such directory is termed a User File Directory (UFD). When a user wishes to use the system, he (or she) is given access to files in a User File Directory by way of a User Identification Code (UIC). The UIC is assigned in one of two ways: either by "signing on" under a UIC in systems with an option built into them called Multi-user Protection (not supported on this system) or by use of the MCR SET/UIC command. A UIC is associated with a terminal if Multi-user Protection is not used, or with a user if Multi-user Protection is used. A UIC allows a user full access to all files in the associated UFD. Limited access rights are allowed to files in different UFDs, depending on the file, the UFD, the UIC, and how the system is built. The four access privileges allowed by the system are read, write, extend (allowing the user to modify the file by making it bigger), and delete. UICs with the value g (called the 'group') between 1-7 are called 'system' UICs and are privileged. The value m is called the 'member' and users whose UIC match the group and member of a UFD are said to be the 'owner' of the UFD. Users whose group matches a UFDs group but whose members differ are said to be in the UFD's group. A UFD whose group and member do not match a given user's UIC and do not belong to a system group are said to belong to the 'world'. These subgroupings (system, owner, group and world) determine

the access privileges that a user has given his UIC and the UFD to which a given file belongs. All files in this system have the following access privileges.

- System UICs (g is between 1 and 7) have read, write, extend, and delete privileges to all files.
- Non-system UICs have read, write, extend and delete access privileges to all files that they own (UIC g and m = UFD g and m).
- Non-system UICs also have read, write, extend and delete access privileges to all files in their group (UIC g = UFD g, but UIC m <> UFD m).
- Non-system UICs only have read access to files in the world (UIC g <> UFD g).

It is possible to have files in a UFD owned by a different UIC. This generally happens when a file from a system UFD is copied to a non-system UFD without transferring file ownership, or a file is created in a non-system UFD by a user or task running under a system UIC. Refer to the PIP manual for a further discussion of this in the description of the /FO switch.

For the CW system seven UFDs have been generated. They are:

- [1,1] (system) Contains system libraries, Fortran libraries and system generation data.
- [1,2] (system) Contains the system boot startup file and text of Fortran and system error messages.
- [1,54] (system) Contains system image, and task images of system tasks like PIP, Fortran, etc.
- [2,2] (system) Contains source and task files of software generated for the CWI system by EG&G, like INPUT, AUTPLT, etc.
- [7,1] (system) Contains source and task files of software generated for the CWII system by EG&G, like FTR, MIS, etc.

- [200,1] (non-system) Contains data files generated by or used by the software system, like the menu files.
- [200,2] (non-system) Contains analyst formatted data and log files generated by the software system.

Tasks are all run under UIC [2,2]. The operator's terminal is nominally set to [2,2].

4-2.1.2 Filenames. Filenames are of the form:

DEV:[g,m]FILENAME.EXT;VER

where

DEV: is the mnemonic of the file structured device where the file is to be accessed. On this system, DEV: is either DL1: or DL0: or a 'pseudo-device' discussed in paragraph 4-2.1.4.

[g,m] is the UFD. Its default is the UIC the user is working under.

FILENAME is the name of the file. It is an up-to-nine character alphanumeric name and may start with a digit. There is no default.

.EXT is the file extension. It is generally used as a file type descriptor. It is an up-to-three alphanumeric character name, or it may be blank. It may or may not have a default, depending upon the context in which the filename is used.

;VER is the version. It is an octal number in the range 1-377. It is used to differentiate between files with the same name and extension. Its default is the highest version that currently exists.

In the CW system, certain default extensions are assumed. These are:

.FLX	Files of this type contain Flecs source code.
.FTN	Files of this type contain Fortran source code. These are usually generated as output of the Flecs processor.
.MAC	Files of this type contain Macro-11 source code.
.OBJ	Files of this type contain object code. These are the output of the Fortran and Macro-11 processors.
.TSK	Files of this type contain task images and are the output of the Task Builder.
.CMD	Files of this type contain commands to MCR or other processors.

4-2.1.3 Logical Units. A Logical Unit is the system construct by which communication to actual devices is done. Tasks write output or read input using logical unit numbers rather than specific device or filenames. The system maintains a table which cross references the logical unit number to the device. This allows the programmer to write the program with device independence, and change devices easily without having to re-compile the program.

The Logical Unit Numbers (LUN) are initially assigned at task build time using the ASG task build command. These assignments can be altered using the system directive ASSIGN LUN or its Fortran equivalents (CALL ASSIGN, OPEN). All LUNs which are to be assigned to a file are generally assigned to the device upon which the file is kept, and calls to the various file handling routines are made to build the File Descriptor Block (FDB) (a data structure maintained in the task space used to control a file) and open the file. In the CW system, this is the method that is used.

4-2.1.4 Pseudo- and Logical-Devices. The operating system allows the users to dynamically re-direct data transfers from one peripheral to another. Such a transfer is effected by the REDIRECT command. Transfers mediated by a REDIRECT behave in the following fashion. A program is written and task built to transfer data to a device, TT0:, via a logical unit. Should TT0: be unavailable (for example, if the device was down), all transfers to TT0: could be REDIRECTED to a different terminal, TT3:, or perhaps a completely different kind of device, LP0: (line printer). The system mediates any special formatting required via the device drivers, and this transfer between devices is, therefore, completely transparent to the user.

To give an added degree of device independence, the system also supports the concepts of logical devices and pseudo-devices. Pseudo-devices have device mnemonics which do not correspond to a given device; rather, they are dynamically assigned by the users to various devices as required. The assignments are made on either a local or global (system-wide) basis. The pseudo-device mnemonics and their meanings are:

SY: The system volume for the users, that is, the volume containing the user files.

LB: System library device, that is, the volume containing the system's file.

CO: Console output device.

CL: Console listing device.

NT: Network communication device (not supported in the CW system).

TI: The user's terminal.

Pseudo-device TI: is always the user's terminal. A task which communicates with device TI: communicates with the terminal upon which the command was entered to start the task. General practice is to task build tasks using pseudo-devices wherever possible, and setting up pseudo-devices (via the REDIRECT and/or ASSIGN commands) to direct traffic to the desired devices at run time.

Along with pseudo-devices, the system supports logical devices. Logical devices are the same as pseudo-devices, except that logical device mnemonics have no default meaning to the system; their meaning and scope are wholly determined by the user. The CW system currently supports one logical device, CD:, the classified data disk volume.

Further explanation on logical devices and pseudo-devices can be found in the RSX-11M Operator's Procedures Manual, Volume 2A. This manual also describes the REDIRECT and ASSIGN commands and their use.

4-2.2 Disk and Disk File Organization

The Data Acquisition subsystem has two disk drives, which are identified to the operating system as DL0: and DL1:, indicated by the '0' and '1' on the drives' 'ready' lights. It is sometimes necessary, due to security constraints, to segregate classified or sensitive data from non-classified data.

The Data Acquisition subsystem uses two logical device names - SY:, which is the system disk, and contains all system files and non-classified data; and CD:, which is the classified data disk. SY: is normally assigned to DL0: and CD: is normally assigned to DL1:. All programs in the Data Acquisition subsystem refer to SY: and CD: instead of the actual device names. This allows the redefinition of disks should a drive fail, without having to re-task-build all the tasks. See paragraph 4-6 for single disk operation particulars.

The BAD utility (see paragraph 5-9 of this manual) is used to declassify disks. This utility was selected because it writes a test pattern in every sector (block) of the disk to see if the pattern can be re-read successfully. This obliterates data previously written on the disk, thereby making it acceptable for declassifying a disk. It also obliterates all system data structures on the disk, including directories, identification blocks, headers, etc. The INI command rebuilds the system data structures, and the UFD command rebuilds the user file

directories. Whereas the system disk contains a number of UFDs, the classified data disk contains only two UFDs, [200,1] and [200,2]. On the classified data disk all threat waveform files generated by the program THRTWV (see Section 2), file INVERS.DAT generated by INVERS (see paragraph 2-4) and the files RAWDATA.TMP, CORECT.DAT, and ACOMP.TMP generated by CORECT (paragraph 2-2) are kept. CORECT.DAT, ACOMP.TMP and INVERS.DAT and overwritten each time a test is performed; whereas the contents of CORECT.DAT are not classified, they may be sensitive and so this file appears on the classified data disk for added integrity. All analyst files are also maintained on this disk. The users of the system are responsible for maintaining the integrity of the data on this disk. It should also be noted that two utilities, AUTPLT (uses the AUTO PLOT capabilities of the HP-2648A terminal to plot data files) and FYLDMP (dumps data files) can access data from this disk if the operator specifies files founds on the CD: disk. The system manager and operator are responsible for security when these utilities are used.

4-2.3 Locally Defined Disk File Structures

Following is a description of the format and structure of the files generated by the software subsystem itself.

4-2.3.1 MENU Files. The 'menu' files are unformatted, direct-access files which contain data used to describe various aspects of the environment of the test. Three files are used to contain this information, named MENU.PRI, MENU.SEC and MENU.ODL. These files reside on the system disk (SY:) in UFD [200,1].

MENU.PRI is the file containing the 'primary' menu. This file contains information necessary for the proper correction and reduction of the raw data supplied by the measurement subsystem. It is a random-access file containing 23 records of 16 bytes. The records each contain one piece of information, so

accessing a given record access a particular parameter. The record asignments for MENU.PRI are:

Record No.	Use
1	Contains the date of the current test in ASCII as 'DD-MMM-YY' where DD and YY are the day and year, respectively; and MMM are the first three letters of the month.
2	Contains the time of the current test in ASCII as 'HH:MM:SS' where HH, MM and SS are the hour (24-hour clock), minute and second, respectively.
3	Contains the type of run being done. Run type if one of five character strings: 'RFSN', 'RCAL', 'TCAL', 'PCAL', or 'TEST'. Any other string is treated by the software the same as a 'TEST'.
4	Contains the up-to-five character filename of the signal probe (the signal probe ID).
5	Contains the up-to-five character filenames of the reference probe (the reference probe ID) or the string 'SYN', indicating the synthesizer is being used in reference channel.
6	Contains the test point ID. This record is only used for annotation.
7	Contains tape number. This entry is of the form NNNN where NNNN is the tape cassette number.

8	Contains the up-to-five character filename of the threat waveform (threat waveform ID).
9	Contains the ASCII representation of gain added to the reference channel in dB.
10	Contains the ASCII representation of gain added to the signal channel in dB.
11	Contains the ASCII representation of the delay added to the signal channel in ns.
12	Contains the ASCII representation of the delay added to the reference channel in ns.
13	Contains the ASCII representation of the network analyzer display reference in dB.
14	Contains the ASCII representation of the delta time for use in building the inverse Fourier transform.
15	Contains the ASCII representation of the threat waveform scaling factor. This string must resemble the input form of a floating point number.
16	Contains the tape file number of the transfer function system calibration run. Format is the same as for record seven.
17	Contains the tape file number of the response function system calibration run. Format is the same as for record seven.

18	Contains hard-copy plot and tape storage enable flags in the first and second bytes, respectively. Bytes set to 'Y' enable the corresponding function.
19	First full word contains, in internal integer format, the calculated amplitude plot centerline in dB.
20	Contains the ASCII representation of the Parseval time values.
21	Contains the ASCII representation of the Parseval frequency value.
22	Contains the ASCII representation of the Parseval ratio.
23	Contains the ASCII representation of the reference sensor calibration reference gain in dB.

The file MENU.SEC contains the secondary menu, which has data that does not figure in the data reduction aspect, but is used for annotation. Each record of this random access file contains 32 bytes which contain only ASCII text. The records are assigned thus:

Records	Use
1	Contains test location.
2	Contains test type.
3	Contains test element.
4	Contains log ID. This is intended as a way to cross-reference a data run with an externally maintained test log.
5	Contains the test engineer's name.
6	Contains the sequence number of up to four digits.
7	Contains a one letter test facility code.

8-9	Contains remarks entered by the operator at end of test. The first 32 characters (bytes) of the remarks are stored in record eight and the remainder appear in record nine.
10	Contains the Mission filename of nine characters + '.MIS'.

The file MENU.ODL contains the ODL menu, which has data that sets up the IEEE-488 bus devices. Each of the 12 records of this random access file contains four bytes which contain only ASCII text. The record assignments for MENU.ODL are:

Record	Use
1	Contains the ID number of the data ODL to turn on and set up.
2	Contains the channel or calibration setting (A, B or C).
3	Contains the channel attenuation setting (0-79) in dB.
4	Contains either 'IN' if the integrator should be placed into the data path or 'OT' if it is to be left out.
5	Contains a 'Y' if V_{in} = 0 is desired for the data ODL, else an 'N'.
6	Contains a 'Y' if the reference ODL is to be turned on, else a 'N'.
7	Contains the reference ODL channel or Calpulse (A, B, or C).
8	Contains the reference ODL attenuation (0-79) in dB.
9	Contains 'IN' if the integrator is to be included in the reference data path, else a 'OT'.
10	Contains a 'Y' if V_{in} = 0 is desired for the reference ODL, else an 'N'.

11	Contains the channel to be selected on the VHF switch A.
12	Contains the channel to be selected on the VHF switch B.

4-2.3.2 Data Files. The software subsystem generates five types of files for storage of test data. Four of these files are deleted and re-allocated before each test, and are therefore available for inspection by stand-alone utilities FYLDMP and AUTPLT between tests only for the previous test. The files are allocated on device CD: in UFD [200,1]. They are unformatted, direct access files and contain data in internal single-precision floating point format. Their filenames are RAWDATA.TMP, CORECT.DAT, INVERS.DAT, and ACOMP.TMP. The fifth type of data file is the files that have been saved for use with the interactive analyst software or written to tape.

RAWDATA.TMP is the raw data spool file. Data from the receiver PCU is obtained by the task INPUT and sent via the Send/Receive Message facility of the operating system to the corrections task, CORECT. CORECT converts the data into the internal single precision format and stores data in RAWDATA.TMP as records of triplets of frequency in Hertz, amplitude in millivolts, and phase in millivolts; one triplet per record.

CORECT.DAT is the file which contains corrected data. CORECT de-spools data from RAWDATA.TMP and applies corrections to it to remove the effect of instrumentation, sensors, etc. This corrected data are then written into CORECT.DAT as records of triplets of frequency in Hertz, amplitude in dB, and phase in degrees (+180); one triplet per record.

INVERS.DAT is the file which contains the results of the inverse Fourier transform. When the inverse task INVERS needs a piece of corrected data to build the inverse transform, the task signals CORECT, which responds by sending a message with the next available piece of corrected data. When all data points are received, INVERS writes the data to INVERS.DAT in the form of records of doublets. Each record contains a time point

in seconds, and a magnitude in units determined by the units of the signal probe used in the test.

ACOMP.TMP is a file used to store composite corrections. To save computation and I/O time, corrections generated by CORECT are saved in ACOMP.TMP during the second cycle of a multicycle test and are read back during the third cycle. Data are stored as triplets of frequency in Hertz, amplitude correction in dB, and a zero (since phase correction does not apply in the third cycle of a test); one triplet per record.

Both RAWDATA.TMP and CORECT.DAT use a special record to denote end of test. This record has a frequency value less than zero. This convention was needed to delimit each cycle of a multicycle test. These files also use the convention of a record with the frequency value equal to zero to denote a deleted frequency or deleted frequency range.

Interactive Analyst and tape data files all have the same format. These files are placed in UFD [200,2] on the classified disk, CD:. They are unformatted, direct-access files and contain a data header and the data points in internal single-precision floating point format.

Data filenames are built by the system to reflect the type of data within. The data filename formats are:

AAAAAXXX.EXT

where AAAAA is the file identifier
XXX is the Julian date the file was created
EXT is the extension that signifies the data state

The file identifier is determined by the test type field in the secondary menu.

Test Type	File Identifier
RFSN	INREF
RCAL	SYSRF
TCAL	SYSTF
PCAL	Five character probe ID from main menu
TEST	Facility code file sequence number from the secondary menu

The three-letter extension will be one of the following depending upon the data state:

EXT	Data State
AMA	Ambient noise measured data
UMA	Pick up noise measured data
CMA	CW measured test data
TCA	Time (transformed) calculated data
TDA	Time (transformed) defined data
PMA	Pulse measured test data
FCA	Frequency calculated data
FDA	Frequency defined data

The data file header contains 57 ASCII records each 12 bytes long. The contents of the data header are:

Record #	Contents	Format
1-2	Byte 1 (of Rec#1) indicates CW file - '1' Byte 2 (of Rec#2) indicates data type '1' - frequency domain '2' - time domain Bytes 3,4, and 5 (of Rec#1) Record # of first data point Presently the remainder of record 1 and record 2 are unused	Char
3	Date of origination	'DD-MMM-YY'
4	Time	'HH:MM:SS'
5-7	File Origin Record 5 - input file 1 for an ANL file contains the value of A	Char

	Record 6 - input file 2, if present - for an ANL file contains the value of B	
	Record 7 - 3 character function code and present date	
8	Test # (Sequence #)	4 char (signed integer)
9	Test Location	char
10	Test Description	char
11	Test Engineer	char
12-16	Test Comments for an ANL file contains values for D, E, F, G, H	char
17	Test Point ID	<=12 char
18	Filename of signal probe	>0, <=9 char
19	Signal gain added in dB	4 char (signed integer)
20	Signal delay added in ns	4 char (signed integer)
21	Threat Wave scaling factor	char (1PE12.5 format)
22	Filename of threat waveform	>0, <=9 char
23	Minimum of the file (first triplet or doublet value)	char (1PE12.5 format)
24	Minimum of the file (second triplet or doublet value)	char (1PE12.5 format)
25	Minimum of the file (third triplet value)	char (1PE12.5 format)
26	Maximums of the file (first triplet or doublet value)	char (1PE12.5 format)
27	Maximums of the file (second triplet or doublet value)	char (1PE12.5 format)
28	Maximums of the file (third triplet value)	char (1PE12.5 format)
29	Type of test	'XCAL' / 'TEST'

30	Filename of reference probe	>0, <= 9 char
31	Reference gain added in dB	4 char (signed integer)
32	Reference delay added in ns	4 char (signed integer)
33	Network analyzer display reference dB	4 char (signed integer)
34	For ANL file contains the value for C	char
35	For ANL file contains the value for J	char
36	Plot format	3 char
37	# points in decade 1	4 char
38	# points in decade 2	4 char
39	# points in decade 3	4 char
40	# points in decade 4	4 char
41	# points in decade 5	4 char
42	# points in decade 6	4 char
43	Multi or single channel test	'Multi'/ 'Single'
44	Tape file ID (4 digits (tape no.), 7 char)	4,7
45	Time domain delta T if applicable -1.0 entered in converted data due to unevenly spaced data	char (1PE12.5 format)
46	Transfer function calibration tape file	4,7
47	Response function calibration tape file	4,7
48	Test Element	char
49	Log ID	char
50	Parseval time value	char (1PE12.5 format)
51	Parseval frequency value	char (1PE12.5

		format)
52	Parseval Ratio	char (1PE12.5 format)
53	Phase unwrap delay time	char (1PE12.5 format)
54-57	Reserved for future use	Blank

The data are written in 12 byte records. Each data record contains either two or three data values depending upon the domain of the data. Frequency domain data records contain a triplet of frequency, magnitude and phase data. Time domain data records have two values per record of time interval and amplitude. The data values are stored in real internal binary format.

4-2.3.3 Calibration Files. In order to correct for the effect of instrumentation and sensors, the transfer functions of these devices must be known. To find these functions, calibrations are performed on the instrumentation and sensors. The data from these calibrations are then stored in calibration files. The data from the cal files are used in the corrections phase of the data reduction.

To correct for the effects of instrumentation, two cal files are used. These files are SYSTF.CAL and SYSRF.CAL. These, and all other cal files, are on volume SY: in UFD [200,1]. SYSTF.CAL is the calibration file for transfer function measurements. This cal file is used for all tests when the receiver PCU 'PLOT FORMAT' thumbwheel is set to 'TFA', 'TFB', or 'TFC', except during the first cycle of a multicycle test. The transfer function cal assumes the presence of a reference measurement and a signal measurement being made simultaneously.

The file SYSRF.CAL is the cal file used to correct response function measurements, which assume that the reference channel has the rf source driving it directly, the result is the response of a test point given the fixed reference level. The

response function file is used for tests when the receiver PCU 'PLOT FORMAT' thumbwheel is set to 'RFA', 'RFB', or 'RFC', and in all tests during cycle one of a multicycle test.

Sensor cal files are used to correct for the effects of sensors. These files have names of XXX.CAL, where XXX is an up-to-nine character name which identifies the sensor. The system assumes that the first character of the sensor identification denotes the units that the sensor measures. Currenty, the following sensor types are supported:

First Letter of Sensor ID	Units Measured
I	Amps
V	Volts
B	Teslas

Should any other character appear as the first character of a sensor ID, no units are assumed. The file UN00.CAL is a special cal file designed to have amplitude and phase of zero. This 'unity sensor' is used primarily for testing. When a sensor ID is entered in response to the menu prompts for reference sensor or signal sensor, the software system appends the extension .CAL to the entered ID and accesses the file.

All cal files are unformatted, direct access files consisting of records each containing one triplet of frequency in Hertz, amplitude in dB, and phase in degrees ($\pm$180 degrees). All data in cal files are in internal single-precision floating point representation, and contain an end-of-file record as the last record. This record contains a triplet consisting of a -1 (in the frequency field; this is the end-of-file mark), the delay in seconds computed when the calibration file was generated (in the amplitude field), and a 0 (in the phase field).

4-2.3.4 Threat Waveform Files. Threat waveform files are used in generation of the inverse Fourier transform. The data in the file is multiplied (in the frequency domain, which is equivalent to convolving in the time domain) with the corrected data to generate the inverse transform. These files are created by the THRTWV stand-alone utility and are placed on device CD: in UFD [200,1]. Threat waveforms have a name of the form XXX.WAV, where XXX is an up-to-nine character filename which is also the waveform identifier. When the waveform ID is entered in response to the menu prompt for the waveform, the software system appends the extension .WAV to the entered ID, and access the file. Threat waveforms are stored in the frequency domain. The file is an unformatted, direct-access file and consists of records containing one triplet each. The triplets are organized as a frequency in Hertz, amplitude in dB, and phase in degrees ($\pm 180^{\circ}$). These files contain two end-of-data records. The first end-of-data record contains a -1 in the frequency field, the value of α in the amplitude field, β in the phase field. The second record contains a -2 in the frequency field, the high-frequency filter cutoff point in the amplitude field and low-frequency filter cutoff point in the phase field. All data in these files are in internal single-precision floating point format.

4-3 MESSAGE STRUCTURES

The software system, as stated earlier, consists of a number of tasks whose only global structure is the RSX-11M Message. A Message is a 13-word block which the sender task fills with data and dispatches to a receiver task. The receiver task gets the 13-word block prefixed with a two word block containing the sending task's name. In the following discussion, only the contents of the 13-word data portion will be discussed.

Messages in the CW system are one of four types, the type being denoted by the second byte of the message. Each type will be discussed separately.

4-3.1 Panel Data Block (PDB) Structures

The Panel Data Block (PDB) contains data on the front panel switch settings of the receiver PCU (also the transmitter PCU as the settings must be identical). The PDB data record is the first record from the PCU and is denoted by an asterisk in the first byte of the record. The record from the PCU is 86 bytes and is organized as follows.

Byte #	Contents
1	'*' - the PDB descriptor
2-4	Plot format setting - ASCII characters which are the same as the characters on the Plot Format thumbwheel.
5-10	Samples per decade - each byte contains a character as defined below. Byte 5 contains the samples/decade for the first decade (1-10 kHz) up through byte 10, which contains the samples/ decade for the 6th decade (100-1000 MHz). The samples/decade code is: A = 0 samples/decade B = 25 samples/decade C = 50 samples/decade D = 100 samples/decade E = 250 samples/decade F = 500 samples/decade G = 1000 samples/decade H = KFD setting
11	Cycle number in ASCII. '0' indicates a single cycle test, whereas '1' - '3' is the cycle of a multicycle test.
12-13	<CR><LF>
14-86	Blank pad.

The software system encodes this information and distributes it to each of the modules. The PDB message format is:

Byte #	Use
1	- Unused -
2	PDB descriptor (=1).
3-5	Plot format setting. Data are encoded the same as in the PCU record.
6-11	Samples/decade. Data are encoded the same as in the PCU record, with byte 6 containing the first decade data and byte 11 containing decade six data.
12	Cycle number in internal byte representation. 0 (numeric value as opposed to the ASCII representation) is the single cycle number, whereas 1-3 is the cycle number of a multicycle test.
13-26	- Unused -

4-3.2 Data Block Structures

Data blocks from the PCU to the PDP-11 arrive in one of three formats. In tests that have phase information (plot format of 'TFA' or 'TFC') the blocks arrive in records containing up to five triplets of frequency, amplitude and phase. In tests that do not contain phase information (plot format of 'TFB', 'RFA', 'RFB', 'RFC'), the data blocks arrive in records containing up to seven pairs of frequency and amplitude. For all tests, the end of a sweep is denoted by a record containing an exclamation point (!) as the second character. All data records are started with a number sign (#) as the first character. The formats are summarized below.

For Tests with Phase Data

Byte #	Use
1	'#' - the data block descriptor.
2-5	A four digit integer in ASCII which is the mantissa of the frequency. A decimal point is assumed between bytes 2 and 3.
6	Power of ten by which the frequency mantissa is multiplied.
7	A plus (+) sign or minus (-) sign.
8-11	Amplitude output of the amplitude DVM. This, along with the sign in byte 7, is the amplitude measured in millivolts. 50 mV = 1 dB.
12	A plus (+) or minus (-) sign.
13-16	Phase output of the phase DVM. This, along with the sign in byte 12, is the phase measured in millivolts. 10 mV = 1°.
17-86	Bytes 2-16 are repeated up to four times. A <CR><LF> sequence follows the last data entry in the record. The PCU will fill all records, except in the case of the last record sent before the end-of-data record. Any extra bytes in the record are blank-padded, and padding occurs after the <CR><LF> sequence.

For Tests Without Phase Data

Byte #	Use
1	'#' - the data block descriptor.
2-5	A four digit integer which is the mantissa of the frequency. A decimal point is assumed between bytes 2 and 3.

6	Power of 10 by which the mantissa in bytes 2-5 is multiplied.
7	A plus (+) or minus (-) sign.
8-11	Amplitude output of the amplitude DVM. This, along with the sign in byte 7, is the amplitude measured, in millivolts. 50 mV = 1 dB.
12-86	Bytes 2-11 are repeated up to six more times. A <CR><LF> sequence follows the last data entry in the record. The PCU will fill all records, except the record just before the end-of-data record. Any extra bytes in a record are blank-padded, and padding occurs after the <CR><LF> sequence.

End-Of-Data Record

Byte #	Use
1	'#' - the data block descriptor.
2	'!' - the end-of-data mark.
3-4	<CR><LF> sequence.
5-86	blank-pads

Data messages also have varied formats. When the data input task INPUT receives a data message, the ASCII characters representing the values are converted into integral integers. The values generated are the mantissa of the frequency, the power of ten, the amplitude in mV, and the phase (if phase data is included) in millivolts. The assumed decimal point is ignored by the INPUT task but is re-inserted into the number by CORECT during the spooling operation. This scheme generates four words of data if phase is included, and three words if phase is suppressed, for each measurement point. Since, in a 13-word message, one word is reserved for the descriptor, there are 12 words available for data. For tests where phase is

included, three data entries will fit in a message, and four data entries will fit in a message if no phase is included. The tasks key on the plot format of the PDB to determine whether phase information is present. Since a full data record from the PCU always requires more than one message to send the data from INPUT to CORECT, the first byte of the message is used to tell CORECT how many data entries are in the message. The format of messages from INPUT to CORECT follows: (all data is in internal integer format unless otherwise noted).

For Tests with Phase Data

Byte #	Use
1	Number of data entries in this message in internal byte format.
2	The data descriptor (=2) in internal byte format.
3-4	Frequency mantissa. The corrections task divides this by 1000 before applying the power of 10 multiplier.
5-6	Power of 10 multiplier.
7-8	Amplitude measurement in millivolts.
9-10	Phase measurement in millivolts.
11-18	Same as bytes 3-10.
19-26	Same as bytes 3-10.

For Tests Without Phase Data

Byte #	Use
1	Number of data entries in this message in internal byte format.
2	The data descriptor (=2) in internal byte format.
3-4	Frequency mantissa. The corrections task divides this by 1000 before applying the power of 10 multiplier.
5-6	Power of 10 multiplier.

7-8	Amplitude measurement in millivolts.
9-14	Same as bytes 3-8.
15-20	Same as bytes 3-8.
21-26	Same as bytes 3-8.

Once the data are corrected by CORECT, the data are then distributed to the tasks CRT (for plotting on the terminal) and INVERS (for use in building an inverse Fourier transform). The message format still uses the data descriptor, but the format is different. All data are in internal single-precision floating point format, and only one point is distributed at a time. Therefore, the data count subfield of the descriptor word in the message is unused in this form of a data message. The format follows: (all entries are in internal single-precision floating point format unless otherwise noted).

Byte #	Use
1	- Unused -
2	The data descriptor (=2) in internal byte format.
3-6	The frequency point in Hz.
7-10	The correct amplitude in dB.
11-14	The correct phase in degrees (set to zero for tests without phase data).
15-18	The composite delay, in seconds. This entry is only sent to the task INVERS, and is used to correct the phase for the inverse transform. It is unused otherwise.
19-26	- Unused -

When the input task (INPUT) senses an end-of-data record from the PCU, it generates a message with a special descriptor and no data. This message is propagated throughout the system. Its format is:

Byte #	Use
1	Set to 1 (internal byte format) when an end-of-data block is sent to the inverse task (INVERS) task from the corrections task (CORECT) if: 1) The test is not one which generates the inverse transform, and 2) The test is a multicycle test.
2	The end-of-data descriptor (=4) in internal byte format.
3-26	- Unused -

4-3.3 Error Status Block (ESB) Structures

The receiver PCU generates a record known as an Error Status Block (ESB) whenever it encounters an error condition for which there is no recovery. Included in the ESB is a code for the error condition. The PDP-11 software also generates error codes in a similar fashion when it encounters an error condition from which it cannot recover. These codes can be found in Appendix A of the Operating Manual. The format of the ESB from the PCU is:

Byte #	Use
1	'&' - the Error Status Block descriptor
2-3	Two ASCII digits which are the error code.
4-5	<CR><LF> sequence.
6-86	Blank pads.

When the ESB is sensed and decoded, it is sent in a message to CORECT, which propagates it to the terminal monitor task (CRT), which displays the code in an error message on the CRT display. The format of an ESB message is:

Byte #	Use
1	- Unused -
2	The ESB descriptor (=3) in internal byte format.

3-4	Two ASCII digits which are the error code.
5-26	- Unused -

4-4 GLOBAL FLAGS

Each task in the RSX-11M environment has access to 64 single-bit 'flags'. These flags are numbered 1-64. Flags 1-32 are associated with the task itself; these are called 'local flags'. Local flags can be used to coordinate intratask events, timing, and other such uses. Each task has its own set of local flags. Local flags 25-32 are reserved for system use. Global flags (also called 'common flags') are numbered 33-64 and are fully accessible by any task currently executing. (Global flags 57-64 are reserved for the system's use.) Refer to the RSX-11M Executive Manual for a more in-depth discussion of event flags and some examples of their use.

In the cw system, 11 global flags are defined for use. In addition, two global flags are used for intratask coordination. This is necessary because the 32 global flags are split into two groups of 16 each (flags 33-48, and flags 48-64). The local flags are also split into two groups. The flag manipulation directives only allow operations of multiple flags in a particular group. So a directive to 'WAIT FOR LOGICAL OR' of a number of flags can only wait for flags in the global group; the directive is unable to support a wait for flags in both a global group and a local group at the same time. To remedy this, two global flags were reserved for use as 'local' global flags.

The global flags defined for use in this system are listed below:

Number	Name	Use
33	ABORT	Signals all tasks in the system to go to end-of-task immediately. Set on encountering an error condition from which there is no

		recovery.
34	CRT READY	Set by CRT to indicate that the task CRT is ready to accept a corrected data point for plotting.
35	INVERS READY	Set by INVERS to indicate that the INVERS task is ready to accept a corrected data point for calculation of the inverse Fourier transform.
36	CRT DATA AVAILABLE	Set by CORECT to indicate to CRT that a data message has been sent to CRT containing a data point for plotting.
37	INVERS DATA AVAILABLE	Set by CORECT to indicate to INVERS that a data message has been sent to INVERS containing a data point for inclusion in the inverse Fourier transform calculations.
38	CORECT DATA AVAILABLE	Set by INPUT to indicate to CORECT that a data message has been sent to CORECT containing raw data from the PCU for spooling.
39	PRIMARY MENU AVAILABLE	Set by CRT to indicate to CORECT and INVERS that the primary menu has been approved by the operator and that the file MENU.PRI containing the primary menu data is available for data extraction.
40	CORECT CLEAR	Set by CORECT to indicate to INVERS, STRTTP, and NHCPLT that CORECT has completed its processing, and that CORECT.DAT, the file containing the corrected data, is available for data extraction.
41	CRT CLEAR	Set by CRT to indicate that the CRT task has completed processing. This

		flag, along with CORECT CLEAR (#40) is used by INVERS to tell when to start the tasks NHCPLT and STRTTP.
43	QLOOK	Set by CRT if the inverse transform plot is to be displayed on the terminal.
45	INVCMP	Set by INVERS to signify to CRT that INVERS is running.
47	PCU READ COMPLETE	Set by the system when the QIO directive which INPUT issues to read data from the PCU completes. This is a 'local' global flag and is only used by INPUT.
48	RAW DATA WAITING	Set by CORECT when raw data are spooled for corrections and remains set until the raw data spool file is depleted. This is a 'local' global flag and is only used by CORECT.

4-5 OTHER SOFTWARE SYSTEM STRUCTURES

4-5.1 Output Cassette File Structures

The MFE-5450 cassette drive and associated tape cassette are used for storage of test data. A record on the cassette consists of 86 data bytes (and two bytes of CRC validity code which are not available to the user). This length is hard-wired into the drive. The first two records on the tape are directory records. The directory structure can control up to eight data files per side of a cassette. There is room for 2000 records on a cassette (including the directory records). The two directory records are followed by data records which are separated logically into files. Each file has two parts: a "header" in which text data describing the test is kept, and a data section.

The directory records contain ASCII data (as do all tape records). The first six characters of each directory record contain the tape number. This tape number only applies

to one side of a cassette. Following the tape number are four directory entries. Each directory entry contains 17 bytes of information. The 17 bytes are used thus:

Byte #	Use
1-12	The up-to-eight character filename followed by a period and the file extension (filename.ext).
13	An equal sign (=).
14-17	The number of records the file occupies. This number is always four digits long.

The first record's directory entries are filled before the second record's directory entries are accessed.

The Control records delimit the data files. Each Control record starts with a percent sign (%) as the first byte of the record, followed by a 0 or a 1. This descriptor indicates the nature of the Control record. The records used are of the following form:

Record	Meaning
%0 filename.ext=xxxx	'filename' is the up-to-eight character filename followed by the extension. 'xxxx' is the number of records in the file. This record starts the file and identifies the file. Following this record is the 57 record header.
%1 END OF FILE	This record is always the last record of a file. It is the end-of-file marker.

The 57 record header contains data from the PDB, the primary menu, and the secondary menu. The header immediately follows the %1 control record. See paragraph 4-2.3.2 for a description of the header record contents.

The data records follow the header records. Data records consist of six ASCII entries. Each entry is a Fortran E-format floating point number. Data is written to tape using a Fortran FORMAT of 6(X,1PE13.6), yielding 84 bytes of data. Bytes 85 and 86 contain a <CR><LF> sequence. Data records for ambient noise, test data, and pickup noise contain two triplets each. Each triplet contains a frequency in Hertz, amplitude in dB, and phase in degrees between -180 and +180 . Should an odd number of triplets result from a test, the last record is blank padded to byte 84. Data records for the inverse transform contain up to three pairs. Each pair contains time point in seconds, and a magnitude in the units of the signal probe. The last record contains only two doublets (as there are always 512 data points in the inverse transform) and is blank padded to byte 84.

4-5.2 Frequency Table Entries

The PCUs contain firmware tables which contain the mantissas of the frequencies to be generated for a test. These mantissas are then scaled to the proper decade and these data are fed via the GPIB to the frequency synthesizers. It is necessary to match these table entries whenever a program interacts with the PCU, as with the Amplitude PROM program AMPL and Delete Frequency PROM program DEL. To do this, a table of mantissas can be generated by the formula

This formula, when coded in PDP-11 Fortran, becomes

```
FTAB(I) = DINIT ((1.DO * (DFLOAT (I-1) ** 1.D-3) * 1000)
```

where FTAB(I) is the ith frequency table entry (i ranges from 1 to 1000), DFLOAT is the integer to double precision floating point conversion, and DNINT is the double precision nearest integer conversion. The result is an integer (in floating point representation) in the range 1000 to 9977. When less than 1000 points per decade are selected for a given decade, the PCU

selects frequencies from the table in evenly spaced intervals, starting with the first table entry.

In scaling table entries for a given decade, low frequencies will have their least significant digits truncated. This occurs because the synthesizers' resolution is only 100 Hz. A frequency of 2626 Hz, for example, when passed to the synthesizer, is truncated to 2600 Hz because the synthesizer cannot resolve a frequency closer than 100 Hz. Due to this truncation, duplicate frequencies may be generated in the low frequency decades. The software system only accepts data from the first of any duplicate frequencies.

4-6 1-DISK OPERATION

The system is designed to operate with only one of the two disk drives functioning. The disk drive which is functioning must have the system disk installed in it and must be unit DL0: (refer to paragraph 2-3.1.2 of the Operating Manual, the UNIT SELECT subsection, to determine how to set a drive unit number). When the system is booted, the operator enters the command

@1DISK<CR>

to establish single-disk operation. This causes volume CD: to be assigned to DL0: (along with SY:).

IMPORTANT

CLASSIFIED DATA MUST NOT BE ACQUIRED, NOR CAN CLASSIFIED THREAT WAVEFORM FILES BE BUILT, WHILE IN A SINGLE-DISK CONFIGURATION.

When returning from a single-disk configuration to a two-disk configuration, the data files from the system disk must be erased and the CD: volume reassigned to unit DL1:. The operator effects these operations by entering

@2DISK<CR>

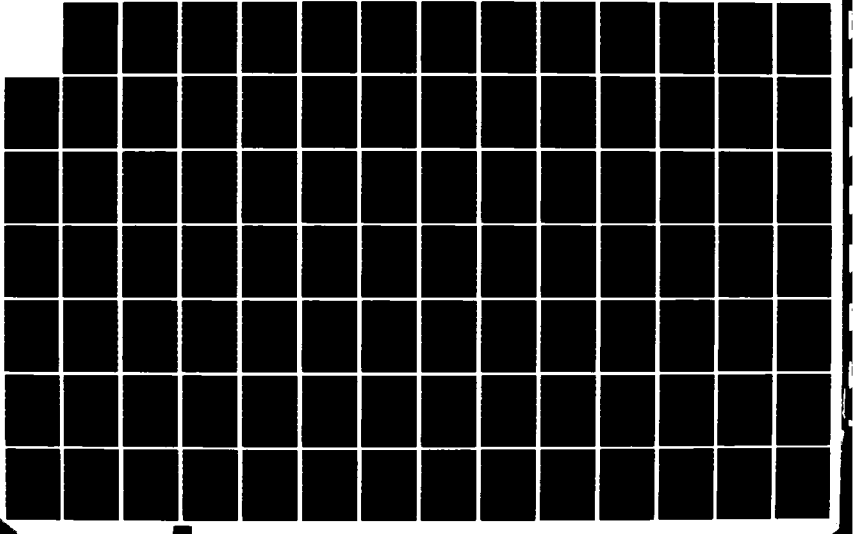
4-7 TERMINAL PORT ASSIGNMENTS

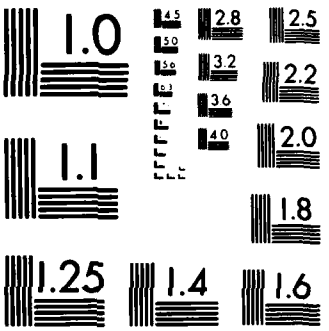
The back panel of the PDP-11 cabinet contains five male EIA RS-282 connectors, each of which is labelled with the type of peripheral to be connected to it. Each port in the back of the cabinet has a device mnemonic of TTnn:, where nn is the unit number (leading 0s can be suppressed). The colon (:) is required syntax indicating a device name. The device mnemonics are assigned to the peripheral ports in the following manner:

Mnemonic	Peripheral Port
TT0:	GRAPHICS TERM HP HP-2648A
TT1:	RCVR PCU
TT2:	TAPE UNIT MFE-5450
TT3:	PROM PROGRAMMER PRO-LOG M900
TT4:	PLOTTER TEK-4662

AD-A151 622 CW MEASUREMENT SYSTEM SOFTWARE SYSTEM MAINTENANCE MANUAL(U) EG AND G WASHINGTON ANALYTICAL SERVICES CENTER INC ALBUQUERQU.. R NELSON ET AL. 02 APR 82 2/3

UNCLASSIFIED EG/G-AG-1435 DNA-6232F DNA001-80-C-0290 NL





SECTION 5
OPERATING SYSTEM SUPPORT PROGRAMS

5-1 GENERAL

The software supplied by Digital Equipment Corporation for use with the PDP 11-34 computer includes the Operating System (RSX-11M); an editor (EDT) for manipulation of text files; a Fortran compiler (F4P); a "task builder" (TKB), which is DEC's name for their linking loader; the Peripheral Interchange Program (PIP), which is the programmer's primary file maintenance utility; the Monitor Console Routine (MCR), which controls operator/operating system communication; and the Dump utility (DMP), which provides a way to display data files in various formats.

5-2 RSX-11M

RSX-11M is a real-time multi-programming operating system designed for fast response to external interrupts and with numerous functions to facilitate intertask communication and control. Refer to Digital Equipment Corporation Manuals RSX-11M Beginner's Guide, Introduction to RSX-11M, and RSX-11M Executive Reference Manual. This system uses version 3.2 of RSX-11M.

Intertask communication is accomplished by means of global event flags, shared data files, and send/receive directives. The global event flags used by the software system are listed in Section 4 of this manual.

Refer to the RSX-11M Executive Reference Manual, Chapter 2, for a discussion of event flags.

Shared data files are files which are accessible to more than one program. The files the software system uses are described in Section 4 of this manual. Refer to RSX-11M I/O Operations Reference Manual, Chapter 2.

Send-Receive directives pass a 13-word data buffer between tasks. Refer to the RSX-11M Executive Reference Manual and Section 4 of this manual.

Indirect command files are a feature of the RSX-11M operating system. An Indirect Command File is a file that contains the commands the operator would enter from his terminal.

Indirect Command Files are created with the editor (EDT), and have file names of the form NAME.CMD, where NAME can be any valid filename. They are executed by typing @NAME<CR> on the operator's terminal; the system will access the file and execute the commands therein. Refer to RSX-11M Operator's Procedures Manual and Appendix A of this manual.

5-3 THE EDITOR (EDT)

EDT is the Digital Equipment Corporation trans- operating system editor. It allows creation and modification of Flecs, Fortran, or Macro-11 source files. Commands are provided to find, modify, insert and delete text from pre-existing files, to create new files, save text and transfer text from one file to another. Refer to the DEC Editor Reference Manual.

5-4 FORTRAN IV PLUS (F4P)

The Fortran compiler supplied with the software system is Digital Equipment Corporation's PDP-11 Fortran IV PLUS, which conforms to American National Standard FORTRAN X3.9-1966, with certain enhancements. Refer to the PDP-11 Fortran IV PLUS Language Reference Manual and the IAS/RSX-11 Fortran IV PLUS User's Guide. The Fortran compiler accepts the output of the Flecs pre-processor discussed in Section 6. For details of Flecs operation consult the Flecs User Manual, Appendix C.

5-5 TASK BUILDER (TKB)

The Task Builder links compiled programs with subroutines from the Fortran and other libraries and assembles the whole into a zero-origin task image ready for relocation and execution by RSX-11M. Refer to the RSX-11M Task Builder Reference Manual.

5-6 PERIPHERAL INTERCHANGE PROGRAM (PIP)

The Digital Equipment Corporation's Peripheral Interchange Program (PIP) is the programmer's primary means of file manipulation. With PIP, files can be created, deleted and renamed, transferred from one device, logical unit, file, disk drive, user account, etc., to another; disk space utilization can be checked; and a number of other functions connected with files and input/output devices performed including unlocking 'locked' files. Refer to RSX-11M Utilities Procedures Manual, and Appendix A of this manual.

5-7 MONITOR CONSOLE ROUTINE (MCR)

The Monitor Console Routine monitors the operator's console and communicates information between the operator and the operating system. Refer to the RSX-11M Operator's Procedures Manual.

5-8 FILE DUMP UTILITY PROGRAM (DMP)

This program is used to print out the contents of any file or disk area. The output may be displayed in various formats such as octal or decimal digits or in character format. Refer to the RSX-11M Utilities Procedures Manual.

5-9 THE DISK INTEGRITY CHECKING UTILITY (BAD)

The BAD Utility is used to erase a pack of all data (as in the case of freeing a classified pack). This is done by writing over each and every sector on the disk and checking for any errors that might occur when the sector is re-read. Refer to the RSX-11M Utilities Procedures Manual.

5-10 DISK SAVE AND COMPRESS UTILITY (DSC)

The DSC Utility is used to copy from one FILES-11 medium to another. The utility is distributed in two forms: as a utility which can be run under MCR in a fashion similar to the other utilities, or as a stand-alone system. Refer to the RSX-11M System Procedures Manual (Vol. 2B) for instructions.

SECTION 6
SOFTWARE DEVELOPMENT

6-1 SYSTEM PROGRAM DESIGN LANGUAGE (PDL)

The system design is written in a Program Design Language (PDL) which is an English description using structured programming concepts/constructs of the logical program flow and conditions of program execution.

Program design languages are used to facilitate the design, development and implementation of structured software. The advantages of structured design include clarity, brevity, modularity, self-documentation and ease of maintenance.

There are two levels of program design language (PDLs): a high level (HPDL) and a low level (LPDL).

A system is first outlined at a high level. The basic structure of the program is described, including inputs and outputs, the basic logical functions, and the media involved.

Next, using the HPDL listing as a template, a low level outline is produced which includes and identifies data types and structures, variables, procedures and error handling. If any logical difficulties are encountered at this stage, the programmer returns to the HPDL stage, corrects the difficulty, then re-enters the LDPL effort. This level of PDL appears in the listing manual.

There are five basic logical constructs associated with structured programming. These will be briefly described here and more fully illustrated, with flowchart examples, later. They are:

The If-Then-Else construct, in which one of two logical paths is chosen depending upon a decision made upon entering the block of code;

The Repeat-Until construct, in which a block of code is repeated until a logical condition at the end of the block is satisfied;

The While-Do construct, in which a block of code is repeated while a logical condition at the beginning of the block is satisfied;

The For construct, in which a block of code is executed until a specific numeric condition is satisfied, and

The Case construct, in which a single block of code is chosen from a number of such blocks, based on an input variable.

Procedures are delimited by the keywords PROCEDURE and ENDPROC. A procedure may require additional parameters from the calling routine or it may not. Parameters may be global in scope or may be explicitly passed between the procedure and the calling routine. The following are examples of procedures and procedue invocations.

```
PROCEDURE NOWAIT;
     *  statements
     *
ENDPROC;
```

Procedure NOWAIT is invoked by the statement NOWAIT, i.e.,

```
FOR INDEX :=1 TO 2;
     NOWAIT;
ENDFOR;
```

Either there are no parameters for NOWAIT or the parameters are global.

```
PROCEDURE WAIT (HR,MIN,SEC);
     *  statements
     *
ENDPROC;
```

Procedure NOWAIT is invoked by the statement WAIT in the code with explicit parameters, i.e.,

```
WHILE JOB.NOT.DONE DO;
     WAIT (0,5,30)
ENDWHILE;
```

The literals 0, 5, and 30 are passed to procedure WAIT. Variables may also be used.

Programs are delimited by the keywords PROGRAM and END as they are generally in actual code. The PROGRAM and END keywords cause a level of indentation in the PDL listings.

Included at the end of this chapter are examples of a high level and a low level PDL which use all of the basic logical constructs and illustrate the development of a program using PDLs.

This is a list of the symbols used in PDLs and their meanings:

1. Relational Symbols
   - = equals
   - ≥ greater than
   - < less than
   - >= greater than or equal to
   - <= less than or equal to
   - <> not equal to
2. Assignment Symbol
   - = becomes; the variable to the left of this symbol is assigned the value of the expression to the right of it
3. Mathematical Operators
   - + plus
   - − minus
   - * times
   - / divided by
   - ** raised to the power of

6-2 PDL CONSTRUCTS

6-2.1 IF-THEN-ELSE Statement (See Figure 6-1)

```
IF logical.expression THEN;
   statements
ELSE;
   statements
ENDIF:
```

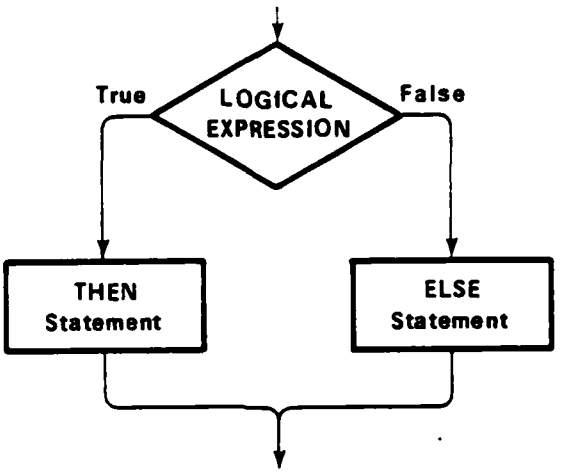


Figure 6-1. Flowchart of the IF-THEN-ELSE Construct

The logical.expression is evaluated upon entering the block. If the expression is true then the statements immediately following the IF-THEN clause are executed; if the expression is false and there is an ELSE clause the statements immediately following the ELSE are executed; otherwise, control is passed to the ENDIF statement. The ELSE clause is optional.

6-2.2 FOR Statement (see Figure 6-2)

```
FOR loop variable := initial value TO [or DOWNTO] final
                     value [BY step];
   statements
ENDFOR;
```

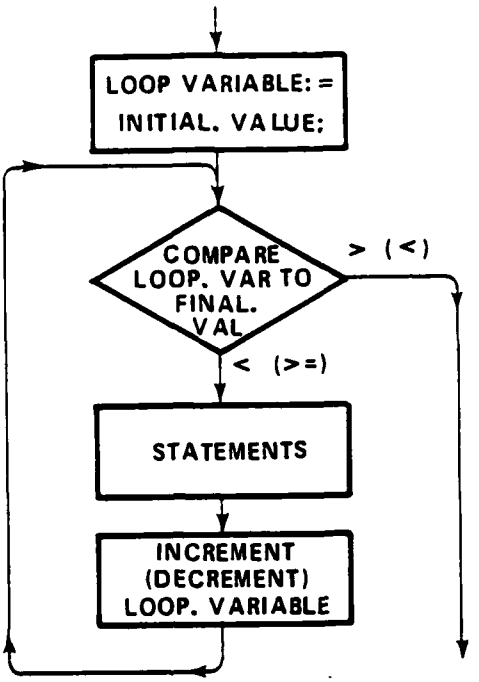


Figure 6-2. Flowchart of FOR Construct

The FOR construct repeatedly executes the contained statements until the value of the loop counter exceeds a limit value. The loop counter is set to the initial value when the block is entered and incremented or decremented (according to whether TO or DOWNTO is used) until it becomes > (< for DOWNTO) the final value. The keyword TO specifies that the loop variable is incremented (by one or by the optional stepsize) each time the loop is executed; the keyword DOWNTO specifies that the loop variable is decremented. Care must be exercised assigning initial and ending values, increment and decrement values and using TO and DOWNTO; it is possible that the loop may never execute or fail to terminate.

6-2.3 REPEAT Statement (see Figure 6-3)

```
REPEAT;
   statements
UNTIL logical.expression
```

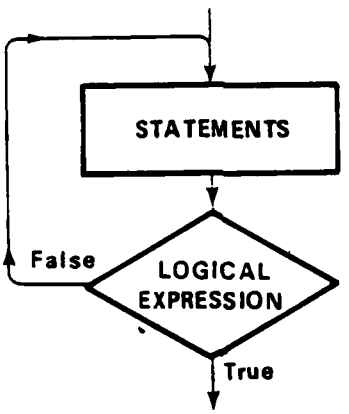


Figure 6-3. Flowchart of REPEAT construct

The REPEAT construct repeatedly executes the contained statements until the logical.expression becomes TRUE. The statements are always executed at least once.

6-2.4 WHILE Statement (see Figure 6-4)

```
WHILE logical.expression DO;
   statements
ENDWHILE;
```

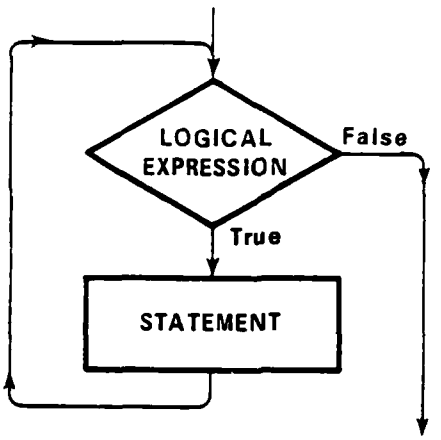


Figure 6-4. Flowchart of WHILE Construct

The WHILE construct repeatedly executes the contained statements as long as the logical.expression remains TRUE. If the logical.expression is FALSE initially; the statements are not executed at all.

6-2.5 CASE Statement (see Figure 6-5)

```
CASE selection.variable OF;
     CASE value (,value);
          statements
     ENDCASE;
     CASE value (,value);
          statements
     ENDCASE;
        *
        *
        *
     ENDCASE;
     OTHERWISE;
          statements
     ENDOTHER;
```

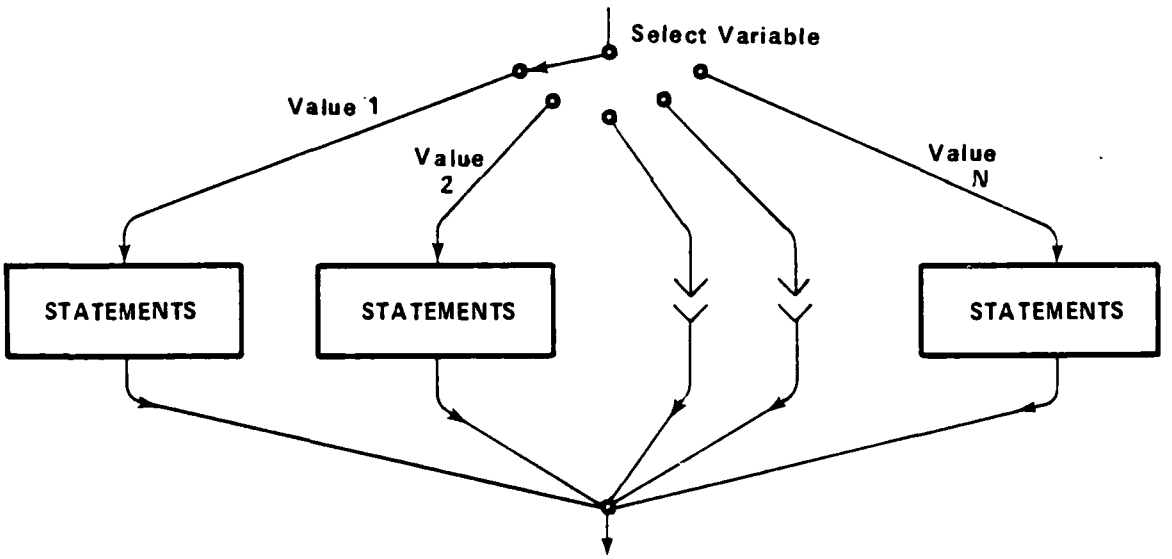


Figure 6-5. Flowchart of CASEOF Construct

The selection.variable is matched against the values specified in each CASE section and if a match is found, the associated statements and no others are executed. If no match is found, then the OTHERWISE statements if present are executed. If no match is found and no OTHERWISE is present, then all statements are skipped.

6-2.6 PROCEDURE Statement (see Figure 6-6)

```
PROCEDURE procname;
    or
PROCEDURE procname (var1,var2,....varn);
    *
    *
    *
ENDPROC;
```

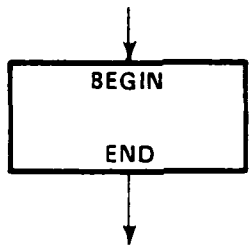


Figure 6-6. Flowchart of PROCEDURE Construct

The procedure is entered and the included statements are executed. When the ENDPROC statement is encountered, control returns to the calling program.

6-2.7 PROGRAM Statement (see Figure 6-7)

```
PROGRAM progname;
    *
    *
    *
END.
```

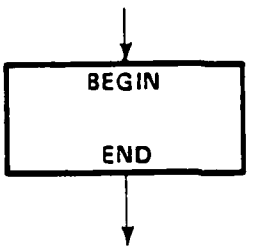


Figure 6-7. Flowchart of PROGRAM Construct

The program is entered and the included statements are executed. When the END. statement is encountered, execution terminates.

6-3 PDL UTILITY PROGRAMS

The Program Design Language (PDL) used by EG&G to design, develop and document computer programs is based on block-structured programming languages such as ALGOL and PASCAL. Two utility programs are used to facilitate formatting and listing PDL programs.

The term "block-structured" means that a program is written as a sequence of logical "blocks", each having only one entrance and one exit. The program is built of these blocks; the interconnection of the various blocks is the "structure" of the program.

Program structure may be illustrated at the listing level by making the logical blocks of which the program is constructed visually distinct from each other. This is done by indenting the beginning of each source statement inside the logical block a certain number of spaces relative to the beginning of the lines preceding the block; thus nested logical blocks present a "stair-step" appearance in the left margin of the listing. Indentation is controlled by the language keywords and associated "END" statements; the listing is indented one level for each keyword and de-indented one level for each "END" statement.

The two utility programs mentioned relieve the programmer of the necessity of keeping track of identation levels, provide a formatted and numbered listing with logic nesting levels flagged. The first program does the indentation and produces a new version of the source file. This means that columnar placement of source lines in the original code is irrelevant. The second program numbers the source lines, flags the indentation levels and produces the listing on a specified physical I/O device.

Following this introduction is documentation for the utility programs and an example shown formatted and unformatted (see Figures 6-3 and 6-9).

```
PROGRAM EXAMPLE1.LPD;

* THIS PROGRAM READS A STRING OF UP TO FORTY ASCII CHARACTERS
* INTO A BYTE ARRAY, CALCULATES THE EQUIVALENT HEXADECIMAL REP-
* RESENTATION OF THE STRING AND PRINTS IT IN ASCII.  AN INPUT SE-
* QUENCE OF TWO ESCAPES TERMINATES THE PROGRAM. 'CHARACTERS' IS
* AN ARRAY OF THE HEXADECIMAL NUMERALS 0-F.  'CHARACTER' IS THE
* INDEX INTO THE ARRAY, 'NIBBLE' IS THE TOP OR BOTTOM HALF OF
* A BYTE.
*
REPEAT;
READ FORTH CHARACTERS FROM USER_FILE INTO INPUT_ARRAY;
ESCAPES := ZERO;
INDEX := ONE;
WHILE INDEX <= NUMBER_OF_CHARACTERS_READ AND ESCAPES < TWO DO;
IF INPUT_ARRAY(INDEX) = ESCAPE THEN;
ESCAPES := ESCAPES + ONE;
ELSE;
ESCAPES := ZERO;
ENDIF;
ENDWHILE;
CONVERT-STRING;
WRITE OUTPUT_ARRAY TO OUTPUT_FILE;
UNTIL ESCAPES = TWO;
*
PROCEDURE CONVERT-STRING;
ARRAY_POINTER := ONE;
NIBBLE := TOP;
FOR INDEX := ONE TO NUMBER_OF_CHARACTERS_READ;
FOR I := ONE TO TWO;
CASE NIBBLE OF;
CASE TOP;
CHARACTER := (INPUT_ARRAY(INDEX) AND OCTAL360)/16;
OUTPUT_ARRAY(ARRAY_POINTED) := CHARACTERS(CHARACTER);
NIBBLE := BOTTOM;
ENDCASE;
CASE BOTTOM;
CHARACTER := INPUT_ARRAY(INDEX) AND OCTAL17;
OUTPUT_ARRAY(ARRAY_POINTED) := CHARACTERS(CHARACTER);
NIBBLE := TOP;
ENDCASE;
ENDCASEOF;
OUTPUT_ARRAY_POINTED := OUTPUT_ARRAY_POINTER + ONE;
ENDFOR;
ENDFOR;
ENDPROC;
END EXAMPLE1.LPD;
```

Figure 6-8. Low-Level PDL Example Before Formatting

```
0001 01- PROGRAM EXAMPLE1.LPD;
0002       * THIS PROGRAM READS A STRING OF UP TO FORTY ASCII CHAR-
0003       * ACTERS INTO A BYTE ARRAY, CALCULATES THE EQUIVALENT HEXA-
0004       * DECIMAL REPRESENTATION OF THE STRING AND PRINTS IT IN
0005       * ASCII. AN INPUT SEQUENCE OF TWO ESCAPES TERMINATES THE
0006       * PROGRAM. 'CHARACTERS' IS AN ARRAY OF THE HEXADECIMAL
0007       * NUMERALS 0-F. 'CHARACTER' IS THE INDEX INTO THE ARRAY.
0008       * 'NIBBLE' IS THE TOP OR BOTTOM HALF OF A BYTE.
0009       *
0010 02-   REPEAT;
0011         READ FORTY CHARACTERS FROM USER_FILE INTO INPUT_ARRAY;
0012         ESCAPES := ZERO;
0013         INDEX := ONE;
0014 03-     WHILE INDEX <= NUMBER_OF_CHARACTERS_READ AND ESCAPES<TWO DO;
0015 04-       IF INPUT_ARRAY(INDEX) = ESCAPE THEN;
0016           ESCAPES := ESCAPES + ONE;
0017 04*       ELSE;
0018           ESCAPES := ZERO;
0019 -04       ENDIF;
0020 -03     ENDWHILE;
0021         CONVERT-STRING;
0022         WRITE OUTPUT_ARRAY TO OUTPUT_FILE;
0023 -02   UNTIL ESCAPES = TWO;
0024       *
0025 02-   PROCEDURE CONVERT-STRING;
0026         ARRAY_POINTER := ONE;
0027         NIBBLE := TOP;
0028 03-     FOR INDEX := ONE TO NUMBER_OF_CHARACTERS_READ;
0029 04-       FOR I := ONE TO TWO;
0030 05-         CASE NIBBLE OF;
0031 06-           CASE TOP;
0032               CHARACTER := (INPUT_ARRAY(INDEX) AND OCTAL360)/16;
0033               OUTPUT_ARRAY(ARRAY_POINTER) := CHARACTERS(CHARACTER);
0034               NIBBLE := BOTTOM;
0035 -06           ENDCASE;
0036 06-           CASE BOTTOM;
0037               CHARACTER := INPUT_ARRAY(INDEX) AND OCTAL17;
0038               OUTPUT_ARRAY(ARRAY_POINTER) := CHARACTERS(CHARACTER);
0039               NIBBLE := TOP;
0040 -06           ENDCASE;
0041 -05         ENDCASEOF;
0042           OUTPUT_ARRAY_POINTER := OUTPUT_ARRAY_POINTER + ONE;
0043 -04       ENDFOR;
0044 -03     ENDFOR;
0045 -02   ENDPROC;
0046 -01 END EXAMPLE1.LPD;
```

Figure 6-9. Low-Level PDL (LPD) Example After Formatting

6-3.1 IPDL - Indent PDLs

This program makes a new version of the same file with all the indenting done automatically (2 columns per indentation). To run the program, type:

```
IPDL (filename)
```

If a filename is specified, the execution of the program begins. If no filename is specified, the program will print a prompt as follows:

```
IPDL>
```

The program will now wait for a filename to be entered. In either case, a default extension of 'PDL' is assumed.

When the program finishes executing, a message

```
ENDING LEVEL = nn
```

is printed. nn is the number of levels unclosed in the file that was input.

No files will be deleted by this program, and a file named the same as the filename entered but with the next highest version number will be created.

6-4 CONVERTING PDLs TO FLECS STATEMENTS

The conversion from PDLs to Flecs is a straightforward process with the restrictions documented below.

In the sections which follow, "condition" denotes an expression which yields a boolean result (i.e., TRUE/FALSE); "statement" denotes a single statement or a series of statements terminated by the Flecs "FIN" statement; brackets "[]" are used to denote optional parameters. Each of the PDL control structures is covered in the sections which follow.

The following restrictions apply:

Flecs must invent many statement numbers in creating the Fortran program. It does so by beginning with a large number (in our implementation 32767) and generating successively smaller numbers as it needs them. Do not use a number which will be generated by the translator. A good rule of thumb is to avoid using five digit statement numbers.

The Flecs translator must generate integer variable names. It does so by using names of the form "Innnn" when nnnn is a five digit number related to a generated statement number. Do not use variables of the form Innnnn and avoid causing them to be declared other than INTEGER. For example, the declaration "IMPLICIT REAL (A-Z)" leads to trouble. Try "IMPLICIT REAL (A-H, J-Z)" instead.

The translator does not recognize continuation lines in the source files. Thus Fortran statements may be continued since the statement and its continuations will be passed through the translator without alteration. However, an extended Flecs statement which requires translation may not be continued. The reasons one might wish to continue a Flecs statement are 1) it is a structured statement or procedure declaration with a one statement scope too long to fit on a line, or 2) it contains an excessively long specification portion, or 3) both of the above. Problem 1) can be avoided by going to the multi-line form. Frequently problem 2) can be avoided when the specification is an expression by assigning the expression to a variable -in a preceding statement and then using the variable as the specification.

In scanning a parenthesized specification, the translator scans from left to right to find the parenthesis which matches the initial left parenthesis of the specification. The translator, however, is ignorant of Fortran syntax including Hollerith parenthese as syntactic parentheses. Thus, avoid placing Hollerith constants containing unbalanced parentheses within specifications. If necessary, assign such constants to a variable, using a DATA or assignment statement, and place the variable in the specification.

Incorrect Example:	Corrected Example:
If (J.EQ.'('	LP = '('
	If (J.EQ.LP)

The Flecs translator will not supply the statement necessary to cause appropriate termination of main and subprograms. Thus, it is necessary to include the appropriate RETURN, STOP,

or CALL EXIT statement prior to the first internal procedure declaration. Failure to do so will result in control entering the scope of the first procedure after leaving the body of the program. Do not place such statements between the procedure declarations and the END statement.

Blanks are meaningful separators in Flecs statements; don't put them in unnecessary places like the middle of identifiers or key words and do use them to separate distinct words like REPEAT and UNTIL.

Let Flecs indent the listing. Start all statements in Column 7 (tab may be used) and the listing will always reveal the true structure of the program (as understood by the translator).

As far as the translator is concerned, FORMAT statements are executable Fortran statements, since it doesn't recognize them as extended Flecs statements. Thus, only place FORMAT statements where an executable Fortran statement would be acceptable. Don't put them between the end of a WHEN statement and the beginnning of an ELSE statement. Don't put them between procedure declarations.

Incorrect Examples:

```
    WHEN (FLAG) WRITE (3,30)
30  FORMAT (7H TITLE:)
    ELSE LINE + LINE+1
```

```
    TO WRITE-HEADER
       . PAGE = PAGE+1
       . WRITE(3,40) H,PAGE
       ...FIN
40  FORMAT(70A1,I3)
```

Corrected Examples:

```
    WHEN (FLAG)
    . WRITE (3,30)
30  . FORMAT (7H TITLE:0
    ...FIN
    ELSE LINE = LINE+1
```

```
    TO WRITE-HEADER
      . PAGE = PAGE+1
      . WRITE(3,40) H,PAGE
40    . FORMAT(70A1,I3)
      ...FIN
```

The translator, being simple-minded, recognizes extended Flecs statements by the process of scanning the first identifier on the line. If the identifier is one of the Flecs

keywords, IF, WHEN, UNLESS, FIN, etc., the line is assumed to be a Flecs statement and is treated as such. Thus, The Flecs keywords are reserved and may not be used as variable names. In case of necessity, a variable name, say WHEN, may be slipped past the translator by embedding a blank within it. Thus, "WH EN" will look like "WH" followed by "EN" to the translator which is blank sensitive, but like "WHEN" to the compiler which ignores blanks.

The following is a brief description of the Flecs language. For further information, refer to the Flecs manual, Appendix C of this manual.

Despite the many differences between the appearance of the Flecs code and ANSI Standard Fortran 66, it must be remembered that Flecs is a 'translator' and that the Flecs code is translated into pure FORTRAN code which can be seen by inspecting the .FTN file that Flecs generates for a given program.

6-4.1 IF-THEN-ELSE

The If-Then-Else is provided in two forms for true and false conditional execution, with a separate form for If-Then-Else. To execute when "condition" is true:

```
IF (condition) statement
```

To execute when "condition" is false:

```
IF (.NOT.condition) statement
   or UNLESS (condition) statement
```

Neither of the previous structures supports an ELSE clause. To use an ELSE clause a different structure is required:

```
WHEN (condition) statement
ELSE statement
```

The WHEN-ELSE keywords are required as a pair.

6-4.2 REPEAT-UNTIL

Flecs provides two structures for the REPEAT-UNTIL construct, for true and false conditional exit.

To execute statement(s) until "condition" is false:

```
REPEAT WHILE (condition) statement
```

To execute statement(s) until "condition" is true:

```
REPEAT UNTIL (condition) statement
```

6-4.3 WHILE-DO

For the WHILE-DO construct, Flecs provides a true and false conditional exit.

To execute statement(s) while "condition" is true:

```
WHILE (condition) statement
```

To execute statements while "condition" is false:

```
UNTIL (condition) statement
```

6-4.4 FOR

This construct is similar to a Fortran DO statement and is coded as:

```
DO (Index = start, end, increment) statement
```

where index, start, end, and increment must be integers. If not specified, increment = +1.

For a FOR statement with a DOWNTO clause, a negative increment must be specified.

6-4.5 CASE

The case statement in Flecs is a SELECT statement. This statement works the same as a CASE and allows the user to specify an optional OTHERWISE clause also.

FORMAT:

```
SELECT (expression)
(Case-1) statement
(Case-2) statement
.
.
.
(Case-n) statement
(Otherwise) statement
FIN
```

Each of the cases is compared for equality with "expression". Because of the way that Flecs builds the

structures, it is wise to make "expression" a simple expression by pre-evaluation.

The otherwise clause is optional.

6-4.6 PROCEDURES

Procedures consist of two types: external procedures which may pass parameters, and local procedures which reference local and/or global data structures. The coding for these types of procedures is different.

For external procedures, the CALL statement of Fortran is used.

```
CALL ZAP
  or
CALL TRY (A1, A2, A3, ... An)
```

Where A1, etc., represent passed parameters.

For local procedures with no arguments passed:

```
   Procedure invocation:
      Procedure-name
   Procedure definition:
      TO procedure-name
            .
            .
      statements
            .
            .
```

6-4.6.1 Procedure-names. Procedure-names may be any string of letters, digits, and hyphens (i.e., minus signs) beginning with a letter and containing at least one hyphen. Internal blanks are not allowed. The only restriction on the length of a name is that it may not be continued.

Examples of valid internal procedure names:

```
   INITIALIZE-ARRAYS
   GIVE-WARNING
   SORT-INTO-DESCENDING-ORDER
   INITIATE-PHASE-3
```

6-4.6.2 Procedure Declaration. A procedure declaration consists of the keyword "TO" followed by the procedure name and its scope. (The set of statements comprising the procedure is called its scope.) If the scope consists of a single simple statement, it may be spaced on the same line as the "TO" and procedure name, otherwise the statements of the scope are placed on the following lines and terminated with a FIN statement.

For example:

```
To INITIALIZE-ARRAYS statement
To GIVE-WARNING
   statement
FIN
```

The Flecs translator will not supply the statements necessary to cause appropriate termination of main and sub-programs. Thus, it is necesary to include the appropriate RETURN, STOP, or CALL EXIT statement prior to the first internal procedure declaration. Failure to do so will result in control entering the scope of the first procedure after leaving the body of the program. Do not place such statements between the procedure declarations and the END statement.

All internal procedure declarations must be placed at the end of the program just prior to the END statement. The appearance of the first "TO" statement terminates the body of the program. The translator expects to see nothing but procedure declarations from that point on.

The order of the declarations is not important. Alphabetical by name is an excellent order for programs with a large number of procedures.

Procedure declarations may not be nested. In other words, the scope of a procedure may not contain a procedure declaration. It may, of course, contain executable procedure references.

Any procedure may contain references to any other procedures (excluding itself.

Dynamic recursion of procedure referencing is not permitted.

All program variables within a main or subprogram are global and are accessible to the statements in all procedures declared within that same main or sub-program.

There is no formal mechanism for defining or passing parameters to an internal procedure. When parameter passing is needed, the Fortran function or subroutine subprogram mechanism may be used or the programmer may invent his own parameter passing methods using the global nature of variables over internal procedures.

The Flecs translator separates procedure declarations on the listing by dashed lines.

SECTION 7
CONTINUOUS WAVE MEASUREMENT SYSTEM

7-1 INTRODUCTION

This section describes the operation of the Program Control Unit (PCU) program which is executed by both the Transmitter (XMTR) PCU and the Receiver (RCVR) PCU. It will first describe the general operation of the program, then describe the major sub-routines. The details of these major sub-routines are described by the comments imbedded within the program code. Refer to the PCU listings manual for the code used by the PCUs.

The PCU program is designed to control the instrumentation required to irradiate a target with a spectrum of radio-frequency (RF) signals, acquiring digitized signals which represent the RF phase and amplitude at the target, and transmitting these signals to a PDP-11 for analysis and plotting. The program may also plot the digitized data directly, while transmitting the digitized data to a cassette recorder. Additional description of this hardware may be obtained from the CWMS Operating Manual, AG-1425.

7-2 MAJOR FUNCTIONS

The PCU program performs the task described above by performing several major functions. These functions are:

- Initialization of the hardware and software.
- Configuration of the CW test in response to the front panel switch values set by the operator.
- Synchronization of the XMTR PCU and RCVR PCU.
- Control of the RF generator and the data acquisition instrumentation.
- Format data for transmission to PDP-11 or recorder.
- Perform computations and controls for plotting data.

The implementation of each of these functions is discussed in detail in Section 8.0. However, a few more general comments are in order before that discussion begins.

7-3 PROGRAM ARCHITECTURE

The PCU program is designed to be event driven. That is, the program remains in an idle state, CWOS, until an event (interrupt) occurs. At Power Up, or upon activation of the Reset switch, the PCU is forced to execute the RESET program. The PCU enters the CWOS state from RESET and remains there indefinitely while awaiting an event.

When the correct event occurs, activation of the Plot/Init switch, the PCU will leave the CWOS state, perform a set of tasks and wait in a different state for another particular event to occur. This process is continued until the CW test is completed, at which time the CWOS state is re-entered.

To insure that the proper sequence of events occurs, the PCU sets software flags and enables/disables hardware at certain points in the program.

7-4 OPERATING MODES

At this point, a description of the various operating modes of the CWMS may facilitate the understanding of the CWMS program.

There are three modes of operation for the CWMS. Two of these set up and verify the operation of the instruments. The remaining mode is used to control the instruments and acquire data.

7-4.1 Manual Mode

The Manual Mode of operation permits the use of the front panel controls on each of the CWMS instruments. In this mode, the PCU has no control of the instruments and is unable to acquire data. Coordination of the XMTR and RCVR PCU frequency is done by the operators. This mode may be used to set up the instruments.

Manual operation of the instruments associated with the PCU may be achieved by setting the Plot Format switch to '000'.

The PCU will be in the idle state, CWOS, while the Format switch is set. To begin manual operation, activate the Plot/Init switch. This drives the PCU through the PINIT state, where it is determined that manual operation has been selected, and into the CONFIG state. While in this state, the PCU sets the DVM's into the free running mode, inhibits the pulses from the synchronizer, and places all of the GPIB instruments into local mode. The PCU then waits in this state for an interrupt from the Start switch. When the Start switch is activated, the RF coax relay is closed and the PCU waits for the Stop switch to be depressed. Activation of the Stop switch causes the PCU to open the RF relay and return to the CONFIG state. Here it waits for another interrupt from the Start switch.

Several important characteristics of the manual operation should now be summarized.

- All instruments are controlled by their front panel switches.
- There is no synchronization between transmitte. and receiver.
- The PCU does not read the DVMs.
- The RF frequency is not advanced by the PCU.
- The RF transmitter is on from the time its Start switch is activated until its Stop switch is activated.
- To terminate manual operation, it is necessary to depress the reset switch.

7-4.2 Semi-Automatic Operation

The semi-automatic mode allows the operator to control the rate at which the frequency spectrum is scanned. One frequency is sampled each time the operators depress the Start switch. In this mode the PCU controls the instruments associated with the CWMS, and determines the frequencies to be sampled.

The PCU and its related instruments may be operated in a semi-automatic mode by setting the Format switch to any value other than '000' and setting the Step Mode switch to MANUAL.

The PCU is in the idle state, CWOS, while the switches are being set. Depressing the Plot/Init switch will drive the PCU to the PINIT state where it determines the table of frequencies to be scanned, transmits the switch values to the PDP-11 or plots the grid, enables the Start switch, and enters tne CWTEST state. The PCU will loop between the CWTEST and CWPLOT states waiting for data to be acquired and plotted, and testing for the end of a test cycle. No data is acquired until the Start switch is depressed.

Depression of the Start switch enables the synchronizer to generate a step pulse. This pulse drives the PCU into the SYSTEP state which initiates the data acquisition sequence and sets the RF frequency and amplitude for the rf frequency synthesizer. The data acquisition sequence is controlled by the TIMER which interrupts the PCU from the CWTEST or CWPLOT states.

The TIMER forces the following sequence to occur.

- Upon receipt of the first step pulse of a test cycle, delay·250 ms before controlling the RF relay. If the test is in cycle 1 of a multi-cycle test, the relay is opened, otherwise, it is closed. The RF relay then remains in this state throughout the test cycle.
- After a 550 ms delay from receipt of the step uplse, the DVMs are commanded to convert the analog RF phase and amplitude signals into digital data. At this point the synchronizer pulse is inhibited.
- After a 660 ms delay from receipt of the step pulse, the phase and amplitude data are transferred from the DVMs to the PDP-11, or to the MFE recorder and the plotter buffer.

Each time the Start switch is activated, the sequence of events described above will occur until the last frequency is sampled. If the plotter is connected to the PCU, the phase data

will be plotted after the last frequency of the spectrum has been sampled. If the test consists of multiple cycles, the PCU will then return to the PINIT state to begin the next cycle; otherwise, the idle state is entered.

Some important characteristics of the Semi-Automatic mode of operation are itemized below.

- The transmitter and receiver are not synchronized. To obtain valid data it is necessary for each operator to alternately depress the Start switch, with the transmitter operator always going first.
- Only one frequency is sampled with each depression of the Start switch.
- All of the instruments are under the control of the PCU.
- The front panel switches of the transmitter and the receiver must have the same settings.

7-4.3 Automatic Operation

The automatic mode of operation allows a complete test, consisting of a single cycle or multiple cycles, to be conducted with a minimum of operator intervention. In this mode the PCU controls the instruments associated with the CWMS, determines the frequencies to be sampled, and automatically scans the desired spectrum.

The PCU and its related instruments may be operated in the automatic mode by setting the Format switch to any value other than '000', setting the Step Mode switch to AUTO.

The PCU is in the idle state, CWOS, while the switches are being set. Depressing the Plot/Init switch will drive the PCU to the PINIT state where is determines the mode of operation, initializes the GPIB instruments, creates the table of frequencies to be scanned, transmits the switch values to the PDP-11 or plots the grid, enables the Start switch, and enters the CWTEST state. The PCU will loop between the CWTEST and CWPLOT states waiting for data to be acquired and plotted, and

testing for the end of a test cycle. No data is acquired until the Start switch is depressed.

Depression of the Start switch enables the synchronizer to generate a series of step pulses after the DataChron clocks reach the 'start time' previously set into the clocks at the receiver and transmitter. Each pulse drives the PCU into the SYSTEP state which initiates the data acquisition sequence and sets the RF frequency and amplitude for the rf frequency synthesizers. The data acquisition sequence is controlled by the TIMER which interrupts the PCU from the CWTEST or CWPLOT states.

The TIMER forces the following sequence to occur.

- Upon receipt of the first step pulse of a test cycle, delay 250 ms before controlling the RF relay. If the test is in cycle one of a multi-cycle test, the relay is opened; otherwise, it is closed. The RF relay then remains in this state throughout the test cycle.
- After a 550 ms delay from receipt of the step pulse, the DVMs are commanded to convert the analog RF phase and amplitude signals into digital data.
- After a 660 ms delay from receipt of the step pulse, the phase and amplitude data are transferred from the DVMs to the PDP-11, or to the MFE recorder and the plotter buffer.

Each step pulse forces execution of the sequence of events described above until the last frequency of the spectrum is sampled. If the plotter is connected to the PCU, the phase data will be plotted after the last frequency of the spectrum has been sampled. If the test consists of multiple cycles, the PCU then will return to the PINIT state to begin the next cycle; otherwise, the idle state is entered.

Some important characteristics of the Automatic mode of operation are itemized below.

- If remote start is selected via front panel switches, the transmitter and receiver are synchronized by the DataChron Clocks. Before the Start

switch is depressed, each operator must be the same 'start time' into each DataChron. Depressing the Start switch then enables the synchronizer to generate the step pulses when the 'start time' arrives. The step pulses are not inhibited until the test is completed.

- The front panel switches of the transmitter and reciever must have the same settings.
- All of the instruments are under the control of the PCU.

7-5 SINGLE CYCLE/MULTI-CYCLE TESTS

The CWMS PCU is capable of conducting a single cycle test or a multi-cycle test. This selection is made with the Single/Multi switch on the PCU front panel. In a single cycle test, the Hold switch has no function.

7-5.1 Single Cycle Test

When the single cycle test is selected, the frequency spectrum is scanned one time only. The RF transmitter is active and the RF probe is in place at the unit under test. As the spectrum is scanned the data from the RF probe is acquired and transmitted to the PDP-11 or the MFE recorder. Upon completion of the spectrum, the PCU enters the idle state.

7-5.2 Multi-Cycle Test

When a multi-cycle test is selected, the frequency spectrum is scanned three times. Each of the three scans is termed a cycle. The first cycle determines the ambient noise during the test. During this cycle, the RF probe is in place at the unit under test, but the RF transmitter is not energized. Cycle two measures the RF phase and amplitude at the unit under test with the transimtter energized for each selected frequency of the spectrum. The third cycle measures the pick-up noise of the RF probe when it is not connected to the unit under test.

The transmitter is energized for each frequency selected in the spectrum.

Data is transmitted to the PDP-11 or the MFE recorder during each of the cycles.

During a multi-cycle test, the Hold switch becomes functional. The Hold switch allows the operator to extend the time interval between cycles. Normally, the interval between the end of cycle one and the start of cycle two will be a few seconds, and the interval between cycle two and cycle three will be a few minutes. However, if the operators each depress the Hold switch before the start of the next cycle, the cycle will not be initiated. To facilitate coordination of this action by the operators, the Hold switch may be depressed at any time during cycles one or two. The switch will not be effective until the end of the cycle. If the operators exercise the Hold option, it becomes necessary to enter a new 'start time' on the Datachron Clock for the start of the next cycle if remote start is selected on the front panel.

7-6 SYSTEM CONFIGURATIONS

The CWMS is capable of operating with three different hardware configurations with the same software program. This offers a degree of back-up operation in the event that certain sub-systems fail to operate. The data acquisition capability of each configuration is equal; however, the data reduction and presentation capabilities are reduced in the back-up modes.

7-6.1 Primary Configuration

The primary hardware configuration for the CWMS consist of the PCU, Plotter, and the PDP-11 with its peripherals. The data acquired by the PCU in this configuration is transmitted to the PDP-11 for analysis. Plotting of reduced data is under control of the PDP-11.

7-6.2 Secondary Configuration

If the PDP-11 becomes inoperative, it may be replaced by an MFE cassette tape recorder. The cable between the PCU and the PDP-11 is replaced by a cable between the same PCU connector and the MFE recorder. The PCU will now transmit the acquired data to the recorder. At a later time, when the PDP-11 becomes operational, the recorder may be connected to the PDP-11 on the same connector to which the PCU was tied and the data may be transmitted to the PDP-11.

In this configuration, the Plotter may be disconnected from the PDP-11 and connected to the PCU via the plotter address switches to obtain graphs while the data is being acquired. Plotting may also be controlled by the PDP-11 by setting the plotter switches. The data plotted is then the reduced data from the PDP-11.

7-6.3 Tertiary Configuration

If both the PDP-11 and the MFE recorder are inoperative, data may still be acquired and sent to the plotter. It is necessary to replace the cable from the PCU to the PDP-11 with a jumper connector. This will allow the PCU to send data to the plotter. The plotter must be configured to 'talk' to the PCU by setting the address switches on the back of the plotter as described in the CWMS Operating Manual, paragraph 3-1.4, Table 20.

SECTION 8
DESCRIPTION OF MAJOR ROUTINES

The major functions, outlined in paragraph 7-2, performed by the PCU programs are implemented by four groups of subroutines which will be discussed in the following paragraphs. In addition, a description of the subroutines associated with the ZT-80 GPIB controller will be presented, as they play an important role in the operation of the PCU. The subroutines are grouped as follows: 1) initialization, 2) test configuration, 3) test control, 4) data acquisition and transmission, and 5) ZT-80 controls.

8-1 INITIALIZATION

Initialization of the PCU hardware and software is performed by the RESET routine whenever power is turned on or the Reset switch is depressed. The RESET routine invokes 20 subroutines to initialize the CWMS. When initialization has been completed, the RESET routine passes control to the monitor, CWOS, where the PCU awaits an interrupt from the Plot/Init switch.

8-1.1 Reset Operation

The Reset switch generates a level 0 interrupt which causes the PCU to terminate its current task and execute the RESET routine. The RESET routine performs the following sequence in initializing the PCU hardware and software.

- Set the parallel input and output ports.
- Set the serial link to the PDP-11.
- Set the flags of the measurement block procedure.
- Set the flags of the operating mode procedure.
- Set the flags of the XMTR amplitude control procedure.
- Calculate the delay length after cycle one and cycle two of a multicycle test.
- Open the RF coax relay.
- Inhibit the system step pulses.

- Inhioit the DVMs.
- Set the DVMs to accept an external trigger pulse
- Blank the CRT of the Network Analyzer.
- Set the interval timer hardware and software.
- Set the interrupt manager hardware and software and enable the PCU interrupts.
- Wait 100 ms for the ZT-80 to load its program from ROM.
- Set the ZT-80 flags and put the GPIB instruments into remote operation.
- Determine if the DVMs are on.
- Transfer the delete ROM ID to RAM.
- Set the frequency, amplitude, and mode of each SD 1702 rf synthesizer.
- Command the Plotter pen to the 'home' position.
- Enter the idle state, CWOS.

8-2 TEST CONFIGURATION

In preparing for a CW test, the operators set the front panel controls of the PCUs to provide the functions prescribed for the test. The PCU is instructed to examine the switches, after they are all set, by depressing the Plot/Init switch of each PCU and set the program flags in accordance with the switch values.

8-2.1 PINIT Operation

The Plot/Init switch generates a level two interrupt which takes the PCU from the CWOS state to the PINIT state. If the PINIT routine has already been invoked by a currently running test, the interrupt is ignored and PCU control returns to the interrupted test. Otherwise, PINIT proceeds with configuring the test being requested as follows:

- Determine if a single or multicycle test is to be done and whether the PDP-11 or the MFE recorder will receive the transmitted data.
- Set the DVMs to accept an external trigger pulse.

- Transfer the front panel switch values to the PDP-11 or the MFE recorder.
- If operation is to be manual, then invoke the START routine and loop there waiting for the Start switch and the Stop switch. A Reset is the only exit from this mode of operation.
- If operation is not manual, then generate the table of frequencies to be sampled if this is a single cycle test or it is is cycle one of a multicycle test.
- If the Plotter is on the PCU and a grid is required, then draw the grid and label the axes.
- Command the Plotter pen to the origin corresponding to the plot format selected.
- Enable the Start switch.
- Enter the CWTEST state and await the system step pulse.

8-3 TEST CONTROL

The test is controlled by the CWTEST routine. Controlling the test consists of aborting the test if requested, plotting the magnitude data as it becomes available, plotting the phase data at the end of a test cycle, and preparing for the next cycle of a multicycle test if necessary.

8-3.1 CWTEST Operation

The CWTEST routine controls the cw test by performing the following sequence of operations:

- If the Stop switch has been activated, abort the test in progress and return to the idle state. Otherwise, continue below.
- If there is no data in the FIFO buffer or it contains Plotter control characters, then loop back to beginning.

- If the FIFO contains RF data, then transfer the data to the Plotter buffer, draw it, and loop back to beginning.
- If the FIFO contains an end of cycle flag, then draw the phase data if it is required.
- Command the Plotter pen to the origin.
- Mute the RF transmitter.
- If a multicycle test is in progress, then regenerate the table of frequencies to be sampled, wait for the cycle delay to expire, and test the Hold switch. If the Hold switch is on, disable the step pulses and wait for the Start switch interrupt.
- If another cycle is to be performed, enter the PINIT state; otherwise, enter the idle state.

8-4 DATA ACQUISITION AND TRANSMISSION

The data acquisition and data transmission to the PDP-11 or MFE recorder is handled by the SYSTEP routine. This routine is interrupt driven to acquire RF data, transmit the data, and set the RF synthesizers for the next frequency to be sampled. Normally the PCU will be in the CWTEST or CWPLOT state when the interrupt for SYSTEP occurs.

8-4.1 SYSTEP Operation

A system step pulse from the synchronizer PC card will generate a level 0 interrupt. In response, the PCU executes the following sequence:

- If the test cycle is done, generate the appropriate delay before beginning the next cycle. Otherwise, continue this sequence.
- If this is the first pass through SYSTEP for this test cycle, then enable the data acquisition time, initialize the plotter flags, set the synthesizer for the first frequency and its corresponding amplitude, and return to the interrupted routine. (Refer

to 7-4-2 for a description of the data acquisition timer sequence.)

- If this is not the first pass through SYSTEP for this test cycle, then enable the data acquisition timer, read the DVMs for the RF data from the previous frequency, transmit the data to the PDP-11 and the Plotter, set the synthesizer for the next frequency and amplitude, and return to the interrupted routine.
- If this is the last frequency sampled, then send the last data points to the PDP-11. If this is a single cycle test, disable the system step pulses. Return to the interrupted program.

8-5 GPIB INSTRUMENTS

The General Purpose Interface Bus (GPIB) is used by the PCU to communicate with the Plotter and the Frequency Synthesizers. There is a significant amount of software devoted to these devices so it is appropriate to discuss it here.

8-5.1 PCU/GPIB Electrical Interface

The PCU controls the GPIB instruments through a GPIB controller, the ZT-80. The ZT-80 is a PC card with the same form factor and electronic interface as the SBC 80/20 microcomputer used in the PCU. The ZT-80 also has the electronic interface for the GPIB. The GPIB timing and signal level characteristics are detailed in the IEEE standard #475, which the ZT-80 satisfies.

8-5.2 PCU/GPIB Command Interface

IEEE standard #475 specifies a fundamental set of commands which allow the instruments on the GPIB to control one another and to pass data between one another. This command set to pass data between one another is very basic and quickly becomes tedious to use. The ZT-80 allows the PC to use a more

elegant command set. The PCU commands are translated by the ZT-80 into the basic set to operate the bus.

8-5.3 PCU/ZT-80 Communications

The PCU communicates with the ZT-80 by very abbreviated commands which direct the ZT-80 to execute programs loaded from its user's PROM. Before proceeding, some definitions are needed.

- An INSTRUMENT PROGRAM is a string of ASCII characters which an instrument on the GPIB interprets as a set of commands. For example, the Frequency Synthesizer sets its output to the frequency 100 MHz when it receives the string GN10000000.
- A CHANNEL COMMAND is an instruction which the ZT-80 translates into a set of basic GPIB commands.
- A CHANNEL PROGRAM is a sequence of channel commands. The channel program is created by the user and placed into the ZT-80 user's PROM.
- A DEVICE LIST is a list of the GPIB instruments which are to receive a particular instrument program or data block transfer.
- The PROGRAM ADDRESS TABLE is a list of the starting address of each channel program that the ZT-80 will execute.

The procedure followed in utilizing the ZT-80 is now described. First, an instrument program was designed for each function required of each instrument. These instrument programs reside in the ZT-80 PROM and are loaded into the ZT-80's volatile memory at power up or reset. The instrument programs are defined in the ZTBUF routine.

Second, a channel program was designed for each instrument program. The channel program specified the device list and data or command buffer to the ZT-80 for each instrument program. The channel programs also reside in the ZT-80 PROM and are loaded into the ZT-80 RAM at power up or reset. The channel programs are defined in the ZTCHAN routine. The device lists

associated with the channel program are defined in the DEVLST routine.

Last, the program address table is defined. This table is transferred to the ZT-80 from the PCU PROM at power up or reset. The PAT routine contains this definition. Each channel program has a priority for execution by the ZT-80 which is set by the position of the program in the program address table.

3-5.4 Typical PCU/ZT-80 Interaction

A typical sequence of events between the PCU and the ZT-80 in controlling a GPIB instrument is outlined below.

- The PCU examines the ZT-30 busy/not busy flag and waits for it to be 'not busy'. This is done by the CWAIT routine.
- The PCU transfers the ID number of the channel program to be executed by the ZT-80 and commands the ZT-80 to begin execution.
- The ZT-80 may set an interrupt to the PCU upon completion of the channel program and it will set the 'not busy' flag.

SECTION 9
UTILITY ROUTINES

There are numerous routines contained in the CWMS PCU program which perform dedicated tasks. These routines are utilized by many of the routines included in the four groups described in Section 8. These routines are isolated from the program flow and their function may be understood by examining the program listing of each utility routine. Included in this set of utility routines is a subset used to control the Tektronix plotter. Refer to the PCU software listings manual.

SECTION 10
BRIEF DESCRIPTION OF CWMS PCU ROUTINES

This section contains a brief description of each public routine in the PCU program. The routines are listed in alphabetical order.

ASCBIN ... converts five or less ASCII digits into a corresponding 16 bit binary value.

BCDBIN ... converts a five digit BCD number to a 24 bit binary value.

BCDBNY ... converts five or less packed BCD digits into a corresponding 16 bit binary value.

BINASC ... converts a 16 bit binary value into a corresponding five digit ASCII character string.

DNASC ... converts a 16 bit binary value to a five character ASCII representation.

CKDVM1 ... determines if DVMs are connected to PCU and powered up.

CKMIDS ... checks the STOP input line to determine if a mid-test stop has been requested.

CNVINT ... calculates the number of step pulses required to generate the delays after cycle one and cycle two.

CONFIG ... inputs Format switch and sets program flags accordingly.

CWAIT ... waits for the ZT-80 to become available for a command.

CWPLOT ... transfers MAGNITUDE and PHASE data from FIFO buffer to Plotter.

CWTEST ... supervisor for controlling, plotting, and terminating CW test.

CYCCHK ... examines the status of the single/multicycle switch and determines whether the PDP-11 or MFE recorder are connected to the PCU.

CYCINV ... at the end of a test cycle it checks for a multicycle test and delays the next cycle if the Hold switch is on.

CYCTIM ... provides delay between test cycles of a multicycle test.

DATA1 ... data tables for use by the Plotter in generating labels and grids.

DBMDBW ... converts RF amplitude values from DBM to DBW for the RF synthesizers.

DECADE ... transfers the log grid marks from a table in ROM to a buffer in RAM for the number of active decades on a test.

DECDAT ... sets the frequency buffer header block as a function of the decade switch buffer value.

DELAY ... provides a one millisecond delay via a program loop.

DELAY1 ... provides multiples of one millisecond delay via a program loop.

DELBLK ... inserts into the measurement data block a flag for each single delete frequency or each group of multiple delete frequencies.

DELID ... prints the deleted frequency ROM identification on the Plotter.

DELPT ... sets the frequency delete pointers for the next decade.

DELROM ... data tables containing delete frequencies for test site.

DEVLST ... data table containing addresses of GPIB devices for each ZT-80 CHANNEL PROGRAM.

DIV16 ... refer to MATH routine below.

DVCNVT ... converts the packed BCD DVM data into signed binary data.

DVMASC ... converts the packed BCD data of the DVM buffer into ASCII characters for transmission to the PDP-11.

DVMPOL ... outputs converted DVM data to FIFO buffer or frequency buffer for magnitude or phase data respectively.

DV1INT ... services that interrupt from both DVMs by setting interrupt flags and inputting each DVM measurement.

ERMSG ... causes CW test to stop and passes error message to Plotter.

FDUMP ... moves two bytes of data from the FIFO buffer to CPUs BC register.

FLOAD ... moves two bytes of data into the FIFO buffer from CPUs BC register.

FRET ... sets carry bit to zero, indicating an unsuccessful task execution.

FWRAP ... maintains the input and output pointers of the FIFO.

GENFRQ ... creates a CW test frequency program, for the Frequency Synthesizer, in a RAM buffer.

GENGRD ... creates an INSTRUMENT PROGRAM for the Plotter to draw grids as a function of the Format switch setting.

HLDTST ... examines the status of the Hold switch and sets a flag.

INCK ... corrects the scale pointer for the TFB and RFC grids.

INTIO ... initializes the interrupt hardware and software upon power up or reset.

LABAXS ... creates an INSTRUMENT PROGRAM for the Plotter to label the axis.

LODPLT ... converts binary data from FIFO buffer to five ASCII digits and loads them into buffer for Plotter.

LOGSCL ... data table containing log grid marks for Plotter as a function of the number of decades to be plotted.

LOGTAB ... data table containing 1000 logarithmically spaced values between 1 and 10.

LSL ... this module is included in XLABEL and helps to label the X axis.

MATH ... provides a 16 by 16 multiply and divide operation.

MODSET ... examines the status of the single/multicycle switch and the hold switch, and sets flags accordingly.

MOVE ... moves a block of data from one memory buffer to another.

MPY16 ... refer to MATH routine above.

MSRBKO ... initializes flags and buffers which are required by the MSR*BLK procedure. This procedure is located in the MSR*BLK module.

MSRBLK ... transmits blocks of measurement data to the PDP-11 via the serial link.

MSRDON ... completes transmission of the measurement block at the end of a test cycle.

MTCYCO ... initializes flags associated with multicycle CW tests. This procedure is located in the CYC$CHK module.

MVCTR ... moves characters from ROM to the plot buffer as part of creating an INSTRUMENT PROGRAM for the Plotter.

NOMEN ... prints all annotation on the Plotter.

NXDEC ... increments the log scale table pointer when generating a plot grid.

PACK ... loads the Plotter buffer with the X and Y coordinates along with Plotter control characters.

PARIO ... initializes the parallel input and output ports of the PCU upon power up or reset.

PARIOS ... contains all of the output port control programs.

PAT ... data table containing the starting addresses of all CHANNEL PROGRAMS executed by the ZT-80.

PAUSE ... this routine is not used.

PINIT ... services interrupt from Plot/Init switch by inputting front panel switch values, labelling plots and initializing program flags in preparation for a cw test.

PLOTO ... moves the Plotter pen to the origin which corresponds to the plot format selected by the operator.

PLOTS ... loads the Plotter buffer with control characters and X, Y coordinates for plotting tic marks on the grid.

PNLBLK ... transmits to the PDP-11 the PCU front panel switch values.

PROMP ... commands the Plotter pen to the home position and turns on the prompt light.

RAMBO ... contains random access storage for all public variables used in the CWMS program.

RAM116 ... contains the buffers to store the frequency, amplitude, and phase data while performing a CW test.

REFLV ... prints the reference synthesizer level label on the Plotter.

RESET ... initializes all of the hardware/software upon power up or reset.

RESTOR ... restores the CPU registers at the end of an interrupt service.

RFFRQ ... maintains the current frequency value in the measurement data block for transmission to the PDP-11.

ROTATE ... sends a command to the Plotter to rotate labels 90 degrees.

RSTR ... adjusts the log scale pointer back to the previous value.

SAMPLE ... prints the number of samples per decade on the Plotter.

SERIOO ... initializes the serial communications link from the PCU to the PDP-11. This procedure is located in the SERL10 module.

SETIM1 ... activates timer for controlling RF data acquisition by the DVMs.

SDAMPL ... sets the amplitude of the transmitter frequency synthesizer as a function of frequency.

SDDEC ... calculates the position of the frequency buffer pointer as a function of the exponent of the frequency.

SDAMPO ... initializes flags for the SD*AMPL procedure.

SDFRQ ... sets the output of each Frequency Synthesizer to the next frequency to be sampled.

SDINIT ... initializes the Frequency Synthesizers to cw mode with correct frequency and amplitude.

SDXMTR ... controls the Frequency Synthesizer amplitude thru the ZT-80.

SRET ... indicates successful completion of a task to the calling routine.

SRLINT ... handles the transmission of ASCII character strings to the PDP-11 via the serial link as an interrupt driven service.

STABLK ... transmits status code data blocks to the PDP-11 which normally indicate error conditions in the test.

STAND ... prints all of the standard annotation for a test on the Plotter.

START ... services interrupt from Start switch by closing and opening the RF relay when in MANUAL operation.

STEP ... copies the test frequencies for a decade, exluding the delete frequencies.

STOP ... disables the PCU interrupt manager and halts execution of the program. Only a Reset will resume operation.

STNANT ... routine contains the ASCII strings which constitute the standard annotation printed on a test plot.

STRTEN ... forms the INSTRUMENT PROGRAM which commands the Plotter to move the pen to a specific location.

SUB1 ... converts packed BCD into two ASCII characters.

SWPOLE ... inputs the front panel switch settings.

SYSTEP ... controls data acquisition and data transmission to the PDP-11.

TIMER ... services interrupt from interval timers for controlling the RF relay and the DVM triggering.

TIMERO ... initializes the interval timer hardware and software.

UNPACK ... transmits Plotter coordinates as ASCII values to the Plotter buffer.

WHODAT ... polls the CWMS devices to determine which are connected and powered up.

XINITS ... initializes the Plotter buffer for the origin which corresponds to the plot format selected.

XLABEL ... generates the X axis decade labels as a function of the number of active decades selected by the operator.

XPOSN ... processes frequency buffer control flags, updates frequency pointer, and computes the X axis coordinate.

YLBL ... generate the Y axis labels of the Plotter as a function of the plot format switch setting.

YLNS ... generates the tic marks for the Plotter Y axis.

YPOSN ... moves the phase data from the real time buffer (FRQBUF) to the post test phase buffer (FIFBUF).

ZTBUF ... data table containing GPIB INSTRUMENT PROGRAMS to be loaded into instruments by ZT-80.

ZNOC ... dummy routine which unconditionally sets device flags to indicate the device is active.

ATCHAN ... data table containing CHANNEL PROGRAMS to be executed by the ZT-80.

ZTINT ... services interrupt from the ZT-80 by setting a program flag.

ZT10 ... activates a CHANNEL PROGRAM when the ZT-80 becomes available.

ZTIDO ... initializes the ZT-80 hardware and software upon power up or reset.

ZTMSK ... initializes the ZT-80 interrupt masks.

ZT80 ... appends the Plotter buffer termination characters to the INSTRUMENT PROGRAM.

APPENDIX A
PERIPHERAL INTERCHANGE PROGRAM

The Digital Equipment Corporation (DEC) supplied Peripheral Interchange Program (PIP) is the system maintainer's primary file access and control utility. With it he may:

rename; delete, and purge old versions of files;
transfer data between files and I/O devices;
list file directories;
unlock files; and
spool files.

Usage of the PIP program is covered in the DEC RSX-11 Utilities Procedures Manual; Chapter 2. A useful subset of PIP will be covered below.

Filename Conventions

DEC filenames are specified as

DEV:[UIC]FILENAME.EXT;VERSION

If an ASN command has been executed so that the device on which the file is resident in the system output device (refer to the DEC RSX-11M Operator's Procedures Manual; Chapter 5, ASN) the DEV: may be omitted; otherwise, specify DL0: (DL: is also acceptable) or DL1:. If the terminal [UIC] (user identification code) has been set to the UIC account number of the file directory containing the desired filename, the [UIC] may be omitted; otherwise, the UIC is specified as [BBB,BBB] where B is an octal digit. Refer to the DEC RSX-11M Operator's Procedures Manual, Chapter 5, the 'SET' command. FILENAME is an up-to-nine alphanumeric character string; EXT is an up-to-three alphanumeric character string. Refer to the DEC Utilities Procedure Manual, Section 1.3. The version is an octal number of range 0-377 and will be discussed under File Maintenance.

Example: to copy a file named USER.EXA in account [333,210] on disk drive DL1: to his terminal, the user might specify:

```
PIP TI:=DL1:[333,210]USER.EXA<CR>
```

which would access the file and copy it (provided it was an ASCII file) to his terminal. Note: in this and all subsequent examples <CR> denotes carriage return.

Example: to set the terminal UIC and system output device to the UIC and device of the file, the user would type:

```
ASN DL1:=SY0:<CR>
SET /UIC=[333,210]<CR>
PIP TI:=USER.EXA<CR>
```

File Maintenance

File maintenance generally includes maintaining adequate storage space on the disk packs, deleting unused files and unlocking files which may have become inaccessible due to system malfunction. Whenever a file is created under RSX-11M, a version number of one is assigned to it. Whenever a new file with the same name is created, the previous file is not deleted; the new file is simply assigned a version number one higher than the previously highest-numbered version. This can lead to an accumulation of obsolete files on disk packs; eventually the packs will become full, at which time some of the files must be removed. This condition can be detected by using the PIP /FR switch, and remedied by using the /PU switch.

Example: to find out how much free space is left on a disk pack, type:

```
PIP DEV:/FR<CR>
```

where DEV:=DL1: or DL0:. PIP will respond with the message

```
DEV: HAS AAAA. BLOCKS FREE, BBBB. BLOCKS USED OUT OF 10240
```

where AAAA and BBBB are decimal numbers which total to 10240.

Example: If the operator wishes to delete all old versions of files on DEV:, he may type:

```
PIP DEV:[*,*]*.*/PU<CR>
```

where DEV: is as above. * is the DEC filename "wildcard" specifier, which means "any or all". Caution is urged; programmers sometimes object to having their "obsolete" versions indiscriminately purged.

Example: If the operator wishes to delete all old versions of a specific file in a specific UIC account on a specific device, or any combination of these, he may do so by using appropriate combinations of the ASN, SET, and PIP commands. For instance:

```
PIP DL1:[333,210]USER.EXA/PU<CR>
```

or

```
ASN DL1:=SY:<CR>
SET /UIC=[333,210]<CR>
PIP USER.EXA/PU<CR>
```

will both delete all but the latest version of USER.EXA in account [333,210] on device DL1:.

If a file is no longer used at all, and the operator wishes to delete it entirely from the disk storage medium, he may specify the PIP /DE switch.

Example: The operator wishes to delete the file USER.EXA;21 from UIC account [333,210] on device DL1:; he may type

```
PIP DL1:[333,210]USER.EXA;21/DE<CR>
```

or

```
ASN DL1:=SY0:<CR>
SET /UIC=[333,210]<CR>
PIP USER.EXA;21/DE<CR>
```

In both the above examples the version number (21) is required; if the operator desires to delete all versions he may specify USER.EXA;*. If it is desired to delete all extensions of a particular filename, for instance USER, the operator may specify USER.*;*. If it is desired to delete all files of a particular extension, for instance all .DTA files, *.DTA;* may be specified, and so forth. If the string *.*;* is entered, all files in the specified UIC on the specified disk will be deleted. If is obvious that the delete command must be used with extreme care.

Example: to unlock a locked file:

```
PIP DEV:[UIC]FILENAME.EXT;version/UN
```

where DEV:,[UIC] and ;version are as discussed previously.

A "locked" file is a file whose descriptor block indicates that it has been opened by some program. If the opening program terminates abnormally, this condition will persist even though in fact no program currently has the file open. When the same or another program subsequently attempts to access the file, the request to open will be rejected by the file manager on the grounds that the file is still open; generally an error condition will be reported by the operating system at this time.

Locked files can be detected by use of the PIP /LI command. When the operating system reports that a file is inaccessible, it also gives a brief reason and the FILENAME part of the file specified. Use PIP to list the directory entry for the file; if it is shown locked (an 'L' before the date in the directory entry) use the PIP /UN command to unlock it. If the FILENAME is unknown, simply list the entire directory for the UIC in which the program was running, and look for locked files.

File Utilities

Using PIP, files may be transferred, directories listed, files renamed and files spooled to a list device.

Example: To transfer files from an input file or device to an output file or device:

```
PIP DEV:=DEV: or
PIP DEV:=DEV:[UIC]FILENAME.EXT;VERSION or
PIP DEV:[UIC]FILENAME.EXT=DEV:[UIC]FILENAME.EXT;VERSION
or
PIP DEV:[UIC]FILENAME.EXT=DEV:
```

where the mnemonics are as in the preceding examples. If a filename is specified, DEV:, [UIC] and VERSION are optional. PIP defaults to the currently assigned terminal UIC and system output device; on input PIP accesses the highest numbered version of the specified file; on output PIP creates a version one higher than the previous (if any) version.

Example: To list a directory:

PIP DEV:[UIC]/LI<CR>

will list all directory entries on DEV: in [UIC}. If DEV: or [UIC] are omitted, PIP defaults to the current terminal and system assignments.

Example: To list a file entry in a directory:

PIP DEV:[UIC]FILENAME.EXT;VERSION/LI<CR>

will list the directory entry for filename.ext if the specified version is present on DEV:[UIC]. If DEV: and [UIC] are omitted, PIP defaults to the current system device and terminal UIC. If ;VERSION is omitted, PIP lists the directory entry for the highest version number present; if * is specified for the version, PIP lists all versions.

Example: To rename a file:

PIP DEV:[UIC]FILENAME.EXT;VERSION=DEV:[UIC]FILENAME.EXT;VERSION/RE

where DEV:, [UIC], and ;VERSION default as previously discussed.

Example: To spool a file to the system list device:

PIP DEV:[UIC]FILENAME.EXT;version/SP

The specified file will be copied to the line printer. The spooling to the printer assumes the presence of the task PRT in the active task list. Refer to the RSX-11M Operator's Procedures Manual (Vol. 2A), Section 5.5, the INS command to find out how to install PRT. See the RSX-11M Utilities Procedures Manual (Vol. 2A) Appendix C for a description of PRT.

APPENDIX B

EXAMPLE OF RSX-11M SYSTEM GENERATION

```
 @SYSGEN
>;
>; SYSGEN PART 1          VERSION 03.1
>;
>; COPYRIGHT (C) 1975, 1976, 1977
>; DIGITAL EQUIPMENT CORP., MAYNARD, MASS.  01754
>;
>;       DETERMINE SYSTEM FEATURES AND ASSEMBLE THE EXECUTIVE
>;
>SET /UIC=[200,200]
>;
>; EXPANDED COMMENTS PROVIDE A DESCRIPTION OF EVERY STEP IN THIS
>; SYSGEN COMMAND FILE.  ON THE OTHER HAND, SHORT COMMENTS
>; PROVIDE VIRTUALLY NO EXPLANATORY INFO.
>;
>* DO YOU WANT EXPANDED COMMENTS? [Y/N]:
>* DO YOU HAVE THE SINGLE RK05 DISTRIBUTION KIT? [Y/N]:
>* DO YOU HAVE THE DUAL RK05 DISTRIBUTION KIT? [Y/N]:
>* DO YOU HAVE THE RL01 DISTRIBUTION KIT? [Y/N]:Y
>;
>; YOU MUST MAKE A COPY OF THE RL01 DISTRIBUTION KIT, IF YOU
>; HAVE NOT DONE SO ALREADY.
>;
>* HAVE YOU COPIED THE RL01 DISTRIBUTION KIT? [Y/N]:Y
>* ARE YOU BUILDING AN RSX-11S SYSTEM? [Y/N]:
>* ARE YOU BUILDING A MAPPED SYSTEM? [Y/N]:Y
>* ARE YOU RUNNING ON A MACHINE WITH AT LEAST 24K WORDS? [Y/N]:Y
>;
>* ARE YOU RUNNING ON A MACHINE WITH A LINE PRINTER? [Y/N]:
>* WILL YOUR SYSTEM INCLUDE DSS-11 SUPPORT? [Y/N]:
>* WILL YOUR SYSTEM INCLUDE ICS/ICR-11 SUPPORT? [Y/N]:
>* WILL YOUR SYSTEM INCLUDE UDC-11 SUPPORT? [Y/N]:
>SET /UIC=[1,1]
>PIP [11,20]*.OBJ;*/DE,*.UDC;*,*.ICR;*,*.IDS;*
PIP -- CANNOT FIND DIRECTORY FILE
SY0:[11,20]*.OBJ;*
PIP -- CANNOT FIND DIRECTORY FILE
SY0:[11,20]*.UDC;*
PIP -- CANNOT FIND DIRECTORY FILE
SY0:[11,20]*.ICR;*
PIP -- CANNOT FIND DIRECTORY FILE
SY0:[11,20]*.IDS;*
>PIP [11,24]*.OBJ;*/DE,*.UDC;*,*.ICR;*,*.IDS;*
PIP -- NO SUCH FILE(S)
SY0:[11,24]*.OBJ;*
PIP -- NO SUCH FILE(S)
SY0:[11,24]*.UDC;*
PIP -- NO SUCH FILE(S)
SY0:[11,24]*.ICR;*
PIP -- NO SUCH FILE(S)
SY0:[11,24]*.IDS;*
>PIP [11,30]UUDCOM.LST;*/DE,ICTAB;*,ICOM;*,DSSDI;*,ISCOM;*
PIP -- CANNOT FIND DIRECTORY FILE
SY0:[11,30]UUDCOM.LST;*
PIP -- CANNOT FIND DIRECTORY FILE
SY0:[11,30]ICTAB;*
PIP -- CANNOT FIND DIRECTORY FILE
SY0:[11,30]ICOM;*
PIP -- CANNOT FIND DIRECTORY FILE
SY0:[11,30]DSSDI;*
PIP -- CANNOT FIND DIRECTORY FILE
SY0:[11,30]ISCOM;*
>PIP [11,34]UUDCOM.LST;*/DE,ICTAB;*,ICOM;*,DSSDI;*,ISCOM;*
PIP -- NO SUCH FILE(S)
```

```
SY0:[11,34]JUDCOM.LST;*
PIP -- NO SUCH FILE(S)
SY0:[11,34]ICTAB;*
PIP -- NO SUCH FILE(S)
SY0:[11,34]ICOM;*
PIP -- NO SUCH FILE(S)
SY0:[11,34]DSSDI;*
PIP -- NO SUCH FILE(S)
SY0:[11,34]ISCOM;*
>PIP [11,10]RSXMC.MAC;*/DE,SYSTB;*,ICTAB;*,RSXBLD.TMP;*
PIP -- NO SUCH FILE(S)
SY0:[11,10]ICTAB;*
PIP -- NO SUCH FILE(S)
SY0:[11,10]RSXBLD.TMP;*
>PIP [11,20]RSXASM.CMD;*/DE,RSXBLD;*
PIP -- CANNOT FIND DIRECTORY FILE
SY0:[11,20]RSXASM.CMD;*
PIP -- CANNOT FIND DIRECTORY FILE
SY0:[11,20]RSXBLD;*
>PIP [11,24]RSXASM.CMD;*/DE,RSXBLD;*
PIP -- NO SUCH FILE(S)
SY0:[11,24]RSXASM.CMD;*
PIP -- NO SUCH FILE(S)
SY0:[11,24]RSXBLD;*
>SET /UIC=[11,10]
>;
>; NOW WE BEGIN THE SYSGEN QUERY SECTION TO SELECT THE EXECUTIVE
>; FEATURES AND PERIPHERAL DEVICES WE WANT IN THE NEW SYSTEM.
>;
>* LONG DIALOGUE DESIRED FOR EXECUTIVE/PROCESSOR OPTIONS? [Y/N]:
>* ASSEMBLY LISTING FILES DESIRED? [Y/N]:
>* TASK BUILDER MAP DESIRED? [Y/N]:
>;
>;       BEGIN EXECUTIVE SERVICE OPTIONS.
>;
>* **DO YOU WANT FILES-11 ACP SUPPORT? [Y/N]:Y
>* DO YOU WANT RMS RECORD LOCKING AND PLACEMENT CONTROL SUPPORT? [Y/N]:Y
>* **DO YOU WANT CHECKPOINTING? [Y/N]:Y
>* DO YOU WANT DYNAMIC CHECKPOINT ALLOCATION? [Y/N]:Y
>* DO YOU WANT DYNAMIC MEMORY ALLOCATION SUPPORT? [Y/N]:Y
>* DO YOU WANT AUTOMATIC MEMORY COMPACTION? [Y/N]:Y
>* DO YOU WANT THE MEMORY MANAGEMENT DIRECTIVES? [Y/N]:Y
>* DO YOU WANT THE SEND/RECEIVE BY REFERENCE DIRECTIVES? [Y/N]:Y
>* DO YOU WANT THE GET MAPPING CONTEXT DIRECTIVE? [Y/N]:
>* DO YOU WANT MULTI-USER PROTECTION SUPPORT? [Y/N]:
>* DO YOU WANT ANSI MAGTAPE ACP SUPPORT? [Y/N]:
>* DO YOU WANT SUPPORT FOR ONLINE, USER MODE DIAGNOSTICS? [Y/N]:Y
>* *DO YOU WANT LOADABLE DEVICE DRIVER SUPPORT? [Y/N]:Y
>* DO YOU WANT NETWORK ACP SUPPORT? [Y/N]:
>* DO YOU WANT AST SUPPORT? [Y/N]:Y
>* **DO YOU WANT TASK TERMINATION AND DEVICE NOT READY MESSAGES? [Y/N]:Y
>* DO YOU WANT POWERFAIL RECOVERY? [Y/N]:Y
>* **DO YOU WANT GET PARTITION PARAMETERS DIRECTIVE? [Y/N]:Y
>* DO YOU WANT GET SENSE SWITCHES DIRECTIVE? [Y/N]:
>* **DO YOU WANT GET TASK PARAMETERS DIRECTIVE? [Y/N]:Y
>* DO YOU WANT SEND/RECEIVE DIRECTIVES? [Y/N]:Y
>* DO YOU WANT ALTER PRIORITY DIRECTIVE? [Y/N]:Y
>* DO YOU WANT CONNECT TO INTERRUPT VECTOR DIRECTIVE? [Y/N]:
>* DO YOU WANT EXTEND TASK DIRECTIVE? [Y/N]:Y
>* DO YOU WANT QUEUE I/O SPEED OPTIMIZATIONS? [Y/N]:Y
>* HOW MANY I/O PACKETS SHOULD BE PRE-ALLOCATED? [D R:1.-15.]: 10.
>* ENTER SIZE OF DATA TRANSFER VECTOR IN WORDS [D R:4.-33.]: 8.
>* *DO YOU WANT INSTALL, REQUEST, AND REMOVE ON EXIT SUPPORT? [Y/N]:Y
>* *DO YOU WANT LOGICAL DEVICE ASSIGNMENT SUPPORT? [Y/N]:Y
>* *DO YOU WANT A NULL DEVICE? [Y/N]:Y
>* *DO YOU WANT THE BASELINE TERMINAL DRIVER? [Y/N]:
```

```
>* DO YOU WANT THE USER ORIENTED TERMINAL DRIVER? [Y/N]:Y
>;
>; IF YOU HAVE AN LA120 OR LA180S, YOU MUST ANSWER YES TO THE NEXT
>; QUESTION.
>;
>* DO YOU WANT FORM FEEDS PASSED DIRECTLY TO TERMINALS? [Y/N]:Y
>* DO YOU WANT SUPPORT FOR A 20K EXECUTIVE? [Y/N]:
>* DO YOU WANT EXECUTIVE LEVEL ROUND ROBIN SCHEDULING? [Y/N]:
>* DO YOU WANT EXECUTIVE LEVEL DISK SWAPPING? [Y/N]:
>* ARE YOU PLANNING TO INCLUDE A USER WRITTEN DRIVER? [Y/N]:
>* DO YOU WANT TO INCLUDE THE EXECUTIVE DEBUGGING TOOL? [Y/N]:Y
>* DO YOU WANT REGISTER AND STACK DUMP AT SYSTEM CRASH? [Y/N]:Y
>* ENTER DUMP DEVICE CSR ADDRESS [O R:160000-177700 D:177564]:
>* DO YOU WANT CRASH DUMP ANALYSIS SUPPORT? [Y/N]:
>* DO YOU WANT THE PANIC DUMP ROUTINE? [Y/N]:Y
>* ENTER THE DUMP DEVICE CSR ADDR [O R:160000-177700 D:177514]:
>* DOES YOUR PROCESSOR HAVE A CONSOLE SWITCH REGISTER? [Y/N]:Y
>* DO YOU WANT DEVICE ERRORS AND TIMEOUTS LOGGED? [Y/N]:
>* DO YOU WANT UNDEFINED INTERRUPTS LOGGED? [Y/N]:
>* DO YOU WANT PARITY ERROR TRAPS LOGGED? [Y/N]:
>* **ENTER SIZE OF DYN. STORAGE REGION IN WORDS [D R:256.-16384.]: 256.
>* DO YOU WANT ROTATING PATTERN IN DATA LIGHTS? [Y/N]:
>;
>;      END EXECUTIVE SERVICE OPTIONS.
>;
>;      BEGIN PROCESSOR OPTIONS.
>;
>* IS YOUR TARGET PROCESSOR A PDP-11/70? [Y/N]:
>* ENTER SIZE OF PHYS. MEMORY IN 1024 WORD BLOCKS [D R:16.-124.]: 124.
>* DO YOU WANT FLOATING POINT PROCESSOR SUPPORT? [Y/N]:Y
>* DO YOU HAVE A PROGRAMMABLE CLOCK? [Y/N]:
>* IS YOUR LINE FREQUENCY 50 HERTZ? [Y/N]:
>* DO YOU WANT KW11-Y WATCHDOG TIMER SUPPORT? [Y/N]:
>* DO YOU WANT PARITY MEMORY SUPPORT? [Y/N]:Y
>* DOES THE TARGET PROCESSOR HAVE A CACHE MEMORY? [Y/N]:
>;
>;      END PROCESSOR OPTIONS.
>;
>;      BEGIN PERIPHERAL OPTIONS.
>;
>* EXPANDED COMMENTS DESIRED FOR PERIPHERAL OPTIONS? [Y/N]:
>PIP RSXBLD.TMP;*/DE
PIP -- NO SUCH FILE(S)
SY0:[11,10]RSXBLD.TMP;*
>* DO YOU WANT DISK WRITECHECK SUPPORT? [Y/N]:
>* DO YOU HAVE ANY PROCESS I/O OR LABORATORY PERIPHERALS? [Y/N]:
>* HOW MANY CM/CR11 CARD READERS DO YOU HAVE? [D R:0.-16.]:
>* HOW MANY TA11 DUAL CASSETTES DO YOU HAVE? [D R:0.-16.]:
>* HOW MANY RJ/RWP04-05-06 DISK CONTROLLERS HAVE YOU? [D R:0.-16.]:
>* HOW MANY RF11 DISK CONTROLLERS DO YOU HAVE? [D R:0.-16.]:
>* HOW MANY RK11 DISK CONTROLLERS DO YOU HAVE? [D R:0.-16.]:
>* HOW MANY RL11 DISK CONTROLLERS DO YOU HAVE? [D R:0.-16.]: 1
>* DO YOU WANT THE DRIVER TO BE LOADABLE? [Y/N]:
>* ENTER VECTOR ADDRESS OF THE NEXT RL11 [O R:60-774 D:330]: 160
>* WHAT IS ITS CSR ADDRESS? [O R:160000-177700 D:174400]:
>* HOW MANY DRIVES DOES IT HAVE? [D R:1.-4.]: 2
>* HOW MANY RK611 DISK CONTROLLERS DO YOU HAVE? [D R:0.-16.]:
>* HOW MANY RP11-C/E DISK CONTROLLERS DO YOU HAVE? [D R:0.-16.]:
>* HOW MANY RWM03 DISK CONTROLLERS DO YOU HAVE? [D R:0.-16.]:
>* HOW MANY RJ/RWS03-04 DISK CONTROLLERS DO YOU HAVE? [D R:0.-16.]:
>* HOW MANY TC11 DECTAPE CONTROLLERS DO YOU HAVE? [D R:0.-16.]:
>* HOW MANY RX11 DISK CONTROLLERS DO YOU HAVE? [D R:0.-16.]:
>* DO YOU HAVE A VT11 GRAPHICS DISPLAY SUBSYSTEM? [Y/N]:
>* DO YOU HAVE A VS60 GRAPHICS DISPLAY SUBSYSTEM? [Y/N]:
>* HOW MANY LP/LS/LV11/LA180 LINE PRINTERS DO YOU HAVE? [D R:0.-16.]: 1
>* DO YOU WANT FAST PRINTER SUPPORT? [Y/N]:
```

```
>* DO YOU WANT THE DRIVER TO BE LOADABLE? [Y/N]:Y
>* ENTER VECTOR ADDRESS OF THE NEXT LINE PRINTER [O R:60-774 D:200]:
>* WHAT IS ITS CSR ADDRESS? [O R:160000-177700 D:177514]:
>* IS IT AN LS11, LA180, OR LP11-V/W? [Y/N]:Y
>* IS IT A 132. COLUMN PRINTER? [Y/N]:Y
>* DO YOU WANT 15. SECONDS BETWEEN PRINTER NOT READY MESSAGES? [Y/N]:
>* HOW MANY SECONDS BETWEEN NOT READY MESSAGES [O R:0.-255. D:15.]: 60.
>* HOW MANY TM02/03 MAGTAPE CONTROLLERS HAVE YOU? [O R:0.-16.]:
>* HOW MANY TM/TMA/TMB11 MAGTAPE CONTROLLERS HAVE YOU? [O R:0.-16.]:
>* HOW MANY PC11 PAPER TAPE READER/PUNCHES DO YOU HAVE? [O R:0.-16.]:
>* HOW MANY PR11 PAPER TAPE READERS DO YOU HAVE? [O R:0.-16.]:
>;
>; INCLUDE CONSOLE IN THE ANSWER TO THE NEXT QUESTION.
>;
>* ENTER NUMBER OF DL11 LINE INTERFACES [O R:1.-16.]: 5
>* DO YOU WANT THE DRIVER TO BE LOADABLE? [Y/N]:
>* ENTER VECTOR ADDRESS OF THE NEXT DL11 [O R:60-774 D:60]:
>* WHAT IS ITS CSR ADDRESS? [O R:160000-177700 D:177560]:
>* ENTER VECTOR ADDRESS OF THE NEXT DL11 [O R:60-774]: 300
>* WHAT IS ITS CSR ADDRESS? [O R:160000-177700]: 176500
>* ENTER VECTOR ADDRESS OF THE NEXT DL11 [O R:60-774]: 310
>* WHAT IS ITS CSR ADDRESS? [O R:160000-177700]: 176510
>* ENTER VECTOR ADDRESS OF THE NEXT DL11 [O R:60-774]: 320
>* WHAT IS ITS CSR ADDRESS? [O R:160000-177700]: 176520
>* ENTER VECTOR ADDRESS OF THE NEXT DL11 [O R:60-774]: 330
>* WHAT IS ITS CSR ADDRESS? [O R:160000-177700]: 176530
>* ENTER NUMBER OF DH11 ASYNCHRONOUS LINE MULTIPLEXERS  [O R:0.-16.]:
>* ENTER NUMBER OF DJ11 ASYNCHRONOUS LINE MULTIPLEXERS [O R:0.-16.]:
>* ENTER NUMBER OF DZ11 ASYNCHRONOUS MULTIPLEXERS [O R:0.-16.]:
>* DO YOU HAVE ANY INTER-PROCESSOR COMMUNICATION DEVICES? [Y/N]:
>;
>;       END PERIPHERAL OPTIONS.
>;
>PIP RSXMC.MAC=[11,10]RSXMC0.MAC/AP
>PIP RSXBLD.CMD=RSXBLD.TMP;*
>PIP RSXBLD.TMP;*/DE
>;
>;
>;       END SYSGEN QUERY SECTION
>;
>PIP [11,24]/RE/NV=RSXASM.CMD;*,RSXBLD.CMD;*
>* DID YOU ANSWER THE SYSGEN QUESTIONS TO YOUR SATISFACTION? [Y/N]:Y
>* DO YOU WANT TO EDIT ANY OF THE QUERY SECTION OUTPUT FILES? [Y/N]:
>INS $BIGMAC/PAR=PAR14K
>;
>; ASSEMBLING THE EXECUTIVE TAKES ANYWHERE FROM 20 MINUTES (ON AN 11/70)
>; TO 4 HOURS (ON AN 11/04). TAKE A BREAK.
>;
>SET /UIC=[11,24]
>TIM
13:50:27 15-AUG-79
>MAC @RSXASM
>TIM
14:20:28 15-AUG-79
>REM MAC
>PIP RSX.OBS=*.OBJ
>PIP RSX11M.OBJ/RE=RSX.OBS
>SET /UIC=[200,200]
>SET /UIC=[1,24]
>PIP /NV=[11,24]RSXBLD.CMD,RSX11M.OBJ
>PIP [200,200]/NV/UF=[11,10]RSXMC.MAC
>SET /UIC=[200,200]
>* DO YOU WANT TO GO DIRECTLY TO THE NEXT PHASE OF THIS SYSGEN? [Y/N]:Y
>SET /UIC=[200,200]
>;
>; SYSGEN PART 2          <VERSION 03.1>
```

```
;
; COPYRIGHT (C) 1975, 1976, 1977
;; DIGITAL EQUIPMENT CORP., MAYNARD, MASS.  01754
;;
;;      BUILD THE EXECUTIVE AND ALL REQUIRED TASKS
;;
;;
;; EXPANDED COMMENTS PROVIDE A DESCRIPTION OF EVERY STEP IN THIS
;; SYSGEN2 COMMAND FILE.  ON THE OTHER HAND, SHORT COMMENTS
;; PROVIDE VIRTUALLY NO EXPLANATORY TEXT.
;;
;* DO YOU WANT EXPANDED COMMENTS FOR SYSGEN PART 2? [Y/N]:
;* HAVE YOU ALREADY BUILT THE EXEC? [Y/N]:
;SET /UIC=[1,54]
;PIP PIP.TSK/PU,EDI,TKB,BIGTKB,VMR,BOO
;INS $LBR
;SET /UIC=[1,1]
;* DO YOU WISH TO ADD THE UDC/LPS/ICS/DSS ROUTINES TO SYSLIB? [Y/N]:
;SET /UIC=[1,24]
;REM LBR
;PIP RSXBLD.CMD/PU
;PIP RSX11M.OLB;*/DE
PIP -- NO SUCH FILE(S)
SY0:[1,24]RSX11M.OLB;*
;PIP [200,200]RSXMC.MAC/PU
;INS $LBR
;LBR RSX11M/CR:80.:640.:128.=RSX11M.OBJ
;PIP RSX11M.OBJ/PU       ! DELETE ALL BUT LATEST OBJECT MODULE FILE
;* DO YOU WISH TO ADD USER WRITTEN DRIVER(S) TO THE EXEC? [Y/N]:
;INS $BIGTKB
;SET /UIC=[1,24]
;TKB @RSXBLD
;* DID THE EXECUTIVE BUILD SUCCESSFULLY? [Y/N]:Y
;REM LBR
;* DO YOU HAVE THE MAP ALREADY? [Y/N]:Y
;* DO YOU WANT TO CONTINUE AND BUILD THE TASKS NOW? [Y/N]:Y
;INS $EDI
;SET /UIC=[1,24]
;* DO YOU WISH TO BUILD THE BIG (5.5K) AND FASTER FILE SYSTEM? [Y/N]:
;* DO YOU WANT TO BUILD THE 2.5K MULTI-HEADER FCP? [Y/N]:Y
;* DO YOU NEED TO EDIT THE TASK BUILD COMMAND FILE FOR FCP? [Y/N]:
;* DO YOU WANT TO BUILD THE MULTI-USER VERSION OF MCR? [Y/N]:Y
;; YOU MUST EDIT THE BUILD COMMAND FILE FOR MCR(MU) TO
;; REFLECT THE SIZE OF SYSPAR.
;EDI MCRMUBLD.CMD
[00019 LINES READ IN]
[PAGE    1]
*L PAR
PAR=SYSPAR:0:10000
*C/100/105
PAR=SYSPAR:0:10500
*[X
[EXIT]

;* DO YOU HAVE TO EDIT THE TASK BUILD COMMAND FILE FOR TKTN? [Y/N]:
;;
;; IF THE ALTER-PRIORITY DIRECTIVE WAS NOT INCLUDED IN THE
;; EXECUTIVE, YOU MUST EDIT THE COMMAND FILE FOR TASK AT.
;;
;* DO YOU HAVE TO EDIT ANY FILES FOR MCR TASKS? [Y/N]:
;* DO YOU WANT TO BUILD THE MULTI-USER PROTECTION TASKS? [Y/N]:
;* DO YOU WANT TO BUILD THE ERROR LOGGING TASKS? [Y/N]:
;* DO YOU WANT TO BUILD TASK ACS? [Y/N]:Y
;* DO YOU HAVE TO EDIT TASK ACS'S BUILD COMMAND FILE? [Y/N]:
;* DO YOU WANT TO BUILD TASK PMD? [Y/N]:Y
;* DO YOU HAVE TO EDIT PMD'S BUILD COMMAND FILE? [Y/N]:
```

```
>* DO YOU WANT TO BUILD TASK SHF? [Y/N]:Y
>* DO YOU HAVE TO EDIT TASK SHF'S BUILD COMMAND FILE? [Y/N]:Y
>EDI SHFBLD.CMD
[00015 LINES READ IN]
[PAGE    1]
*C/\/\./CP.
[NO MATCH]
*L /CP
[1,54]SHF/PR/MM/CP/AL/-FP,MP:[1,34]SHF/-SP=[1,24]SHUFL
*C./CP.
[1,54]SHF/PR/MM/AL/-FP,MP:[1,34]SHF/-SP=[1,24]SHUFL
*EX
[EXIT]

>* DO YOU WANT TO BUILD TASK RMDEMO? [Y/N]:
>* DO YOU WANT TO BUILD TASK PRT (PRINT SPOOLER)? [Y/N]:Y
>;
>; THE DEFAULT PRINT SPOOLER WILL ONLY DELETE FILES NAMED
>; LP.LST, LP.MAP, *.PMD, AND *.DMP.  IF YOU WANT THE SPOOLER
>; TO DELETE ALL FILES, YOU MUST EDIT THE BUILD COMMAND FILE.
>;
>* DO YOU HAVE TO EDIT TASK PRT'S BUILD COMMAND FILE? [Y/N]:
>REM EDI
>SET /UIC=[1,24]
>PIP *.CMD/PU
>* DO YOU WANT TO GENERATE THE MAPS FOR THE ABOVE TASKS? [Y/N]:
>ASN NL:=MP:
>TKB @BOOBLD
>TKB @DMOBLD
>TKB @FCPBLD
>TKB @INDBLD
>TKB @INIBLD
>TKB @INSBLD
>TKB @MCRMUBLD
>TKB @MOUBLD
>TKB @SAVBLD
>TKB @TKNBLD
>TKB @UFDBLD
>TKB @LOABLD
>TKB @UNLBLD
>TKB @PMDBLD
>TKB @SHFBLD
>TKB @PRTBLD
>TKB @ACSBLD
>* DO YOU WANT TO BUILD ANY LOADABLE DRIVERS NOW? [Y/N]:Y
>SET /UIC=[200,200]
>;
>; YOU MAY NEED THE TASK BUILD MAPS FOR THE DRIVER(S) FOR
>; YOUR PARTITION LAYOUT.
>;
>* DO YOU WANT THE MAPS? [Y/N]:
>SET /UIC=[1,24]
>;
>; WHEN ALL DRIVERS ARE BUILT, STRIKE CARRIAGE RETURN WHEN ASKED
>; FOR DEVICE MNEMONIC.
>;
>* WHAT IS THE DRIVER 2-CHARACTER DEVICE MNEMONIC [S]: LP
>* WHAT IS THE DRIVER PARTITION NAME [S]: GEN
>TKB @[200,200]LPDRVBLD
>PIP [200,200]LPDRVBLD.CMD;*/DE
>* WHAT IS THE DRIVER 2-CHARACTER DEVICE MNEMONIC [S]:
>SET /UIC=[1,54]
>PIP LPNEW.TSK/NV/RE=LPDRV.TSK
>PIP LPNEW.STB/NV/RE=LPDRV.STB
>REM TKB
>;
```

```
>; IF YOU WISH TO REBUILD THE UTILITY TASKS AND/OR BUILD THE USER MODE
>; DIAGNOSTICS, YOU MUST DO SO AFTER SYSGEN PART 2 HAS BEEN COMPLETED.
>; AT THAT TIME, MOUNT THE DISK LABELED "UTILITY OBJECTS" AND
>; RUN THE INDIRECT COMMAND FILE [200,200]SYSGEN3.CMD
>;
>SET /UIC=[1,54]
>* IS YOUR TARGET SYSTEM LARGER THAN 32K WORDS? [Y/N]:Y
>* DO YOU WISH TO CREATE A LARGER COPY OF YOUR SYSTEM? [Y/N]:Y
>;
>;       NOTE:   SAVED IMAGES ON THE 11/70 ARE CURRENTLY
>;               SUPPORTED ONLY TO A MAXIMUM OF 124K WORDS
>;               (I.E. N=498.). ALSO, THE NUMBER OF BLOCKS
>;               IS ASSUMED TO BE OCTAL. TO SPECIFY A
>;               DECIMAL NUMBER, APPEND A TRAILING DECIMAL POINT.
>;
>;               FOR SINGLE RK05 DISTRIBUTION KITS, THE
>;               MAXIMUM SYSTEM IMAGE SIZE IS 64K (258.
>;               BLOCKS) DUE TO A LACK OF CONTIGUOUS SPACE.
>;
>* ENTER THE NUMBER OF BLOCKS FOR YOUR SYSTEM IMAGE FILE [S]: 130.
>PIP RSX11M.SYS/CO/NV/BL:130.=RSX11M.TSK
>SET /UIC=[1,54]
>* DO YOU WISH TO DELETE THE SYSTEM BACKUP FILE RSX11M.TSK? [Y/N]:
>;
>; YOUR TARGET SYSTEM IS NOW READY TO SET UP PARTITIONS AND INSTALL
>; THE REQUIRED TASKS.  THE PROCEDURE TO FOLLOW ONCE VIRTUAL MCR
>; IS RUNNING IS:
>;       1) EXTEND POOL SPACE TO BASE OF FIRST PARTITION
>;       2) SET UP YOUR PARTITIONS
>;       3) INVOKE THE INDIRECT FILE INSTALL.CMD IN VMR TO
>;          INSTALL THE PRIVILEGED TASKS YOU JUST BUILT
>;       4) LOAD ANY LOADABLE DRIVERS THAT YOU WANT RESIDENT
>;          IN THE SYSTEM IMAGE.  IF THE TERMINAL AND/OR SYSTEM
>;          DISK DRIVERS ARE LOADABLE, THEY MUST BE LOADED USING
>;          VMR.
>;       5) EXIT FROM VIRTUAL MCR AND BOOT IN YOUR TARGET SYSTEM
>;
>* DO YOU WANT A VMR EXAMPLE DISPLAYED? [Y/N]:
>* DO YOU WANT TO EDIT INSTALL.CMD? [Y/N]:
>PIP LPDRV.TSK/RE/NV=LPNEW.TSK
>PIP LPDRV.STB/RE/NV=LPNEW.STB
>INS $BOO;-1
>INS $VMR;-1
>ASN SY:=LB:
>VMR
ENTER FILENAME: RXS\SX\SX11M.SYS
VMR>SET /POOL
POOL=550:260.:00260.
VMR>SET /POOL=700
VMR>SET /MAIN=SYSPAR:700:105:TASK
VMR>SET /MAIN=GEN:1005:6573:SYS
VMR>PAR
LDR    000000 000000 MAIN TASK
SYSPAR 070000 010500 MAIN TASK
GEN    100500 657300 MAIN SYS
VMR>@[200,200]INSTALL
VMR -- CHECKPOINT SPACE TOO SMALL, USING CHECKPOINT FILE
INS SHF          ! INSTALL SHUFFLER
VMR>^Z

>;
>; WHEN THE INDIRECT FILE EXITS, BOOT IN YOUR TARGET SYSTEM,
>; SET THE DATE AND TIME, AND SAVE THE SYSTEM WITH A BOOTSTRAP.
>; WHEN YOUR SYSTEM REBOOTS ITSELF, PURGE THE OLD TASK FILES.
>;
>; E.G.
```

```
;       >BOO RSX11M
>;
>;      RSX11M V03.1 BL22
>;      >TIM 12:00:00 1-JAN-78
>;      >SAV /WB
>;
>; RSX11M V03.1 BL22   64K  MAPPED
>;      >RED DL0:=SY:
>;      >RED DL0:=LB:
>;      >MOU DL0:MAPRL01
>;      >@[1,2]STARTUP
>;      >* PLEASE ENTER TIME AND DATE (HR:MN DD-MMM-YY) [S]:
>;      >TIM 12:01 1-JAN-78
>;      >@ <EOF>
>;      >SET /UIC=[1,54]
>;      >RUN $PIP
>;      PIP>*.*/PU
>;      PIP>^Z
>;      >
>;
>@ <EOF>
>BOO RSX11M
XDT: 22

XDT>
RSX11M V3.1 BL22


>TIM 8:0 8/1/79
>SAV /WB

  RSX-11M V3.1 BL22   64K    MAPPED
>RED DL0:=SY0:
>RED DL0:=LB0:
>MOU DL0:MAPSYS
>@[1,2]STARTUP
>* PLEASE ENTER TIME AND DATE (HR:MN DD-MMM-YY) [S]:
>TIM
08:00:37 01-AUG-79
>@ <EOF>
>SET /UIC=[1,54]
>INS LB:PIP
>POI\IO\IMP\PMIP\PIP
PIP>[*,*]*.*/PU
PIP>^Z

>
```

APPENDIX C

FLEC'S USERS MANUAL

C-1 INTRODUCTION

Fortran contains four basic mechanisms for controlling program flow: CALL/RETURN, IF, DO, and various forms of tne GO TO.

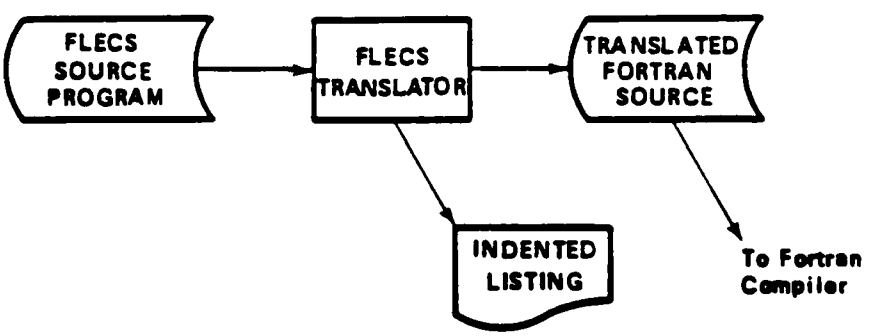
Flecs is a language extension of Fortran which has additional control mechanisms. These mechanisms make it easier to write Fortran by elminating much of the clerical detail associated with constructing Fortran programs. Flecs is also easier to read and comprehend than Fortran.

This manual is intended to be a brief but complete introduction to Flecs. It is not intended to be a primer on Flecs or structured programming. The reader is assumed to be a knowledgeable Fortran programmer.

For programmers to whom transportability of their program is a concern, it should be noted that the Flecs translator source code is in the public domain and is made freely available. The translator was written with transportability in mind and requires little effort to move from one machine to another.

At the University of Oregon, Flecs is implemented on both the PDP-10 and the IBM S/360. The manner of implementation is that of a pre-processor which translates Flecs programs into Fortran programs. The resulting Fortran program is then processed in the usual way. The translator also produces a nicely formatted listing of the Flecs program which graphically presents the control structures used.

The following diagram illustrates the translating process.



C-2 RETENTION OF FORTRAN FEATURES

The Flecs translator examines each statement in the Flecs program to see if it is an extended statement (a statement valid in Flecs but not in Fortran). If it is recognized as an extended statement, the translator generates the corresponding Fortran statements. If, however, the statement is not recognized as an extended statement, the translator assumes it must be a Fortran statement and passes it through unaltered. Thus, the Flecs system does not restrict the use of Fortran statements, it simply provides a set of additional statements which may be used. In particular, GO TOs, arithmetic IFs, CALLs, arithmetic statement functions, and any other Fortran statements, compiler dependent or otherwise, may be used in a Flecs program.

C-3 CORRELATION OF FLECS AND FORTRAN SOURCES

One difficulty of preprocessor systems like Flecs is that error messages which come from the Fortran compiler must be related to the original Flecs source program. This difficulty is reduced by allowing the placement of line numbers (not to be confused with Fortran statement numbers) on Flecs source statements. These line numbers then appear on the listing and in the Fortran source. When an error message is produced by either the Flecs translator or the Fortran compiler, it will include the line number of the offending Flecs source statement, making it easy to locate on the listing.

If the programmer chooses not to supply line numbers, the translator will assign sequential numbers and place them on the listing and in the Fortran source. Thus, errors from the compiler may still be related to the Flecs listing.

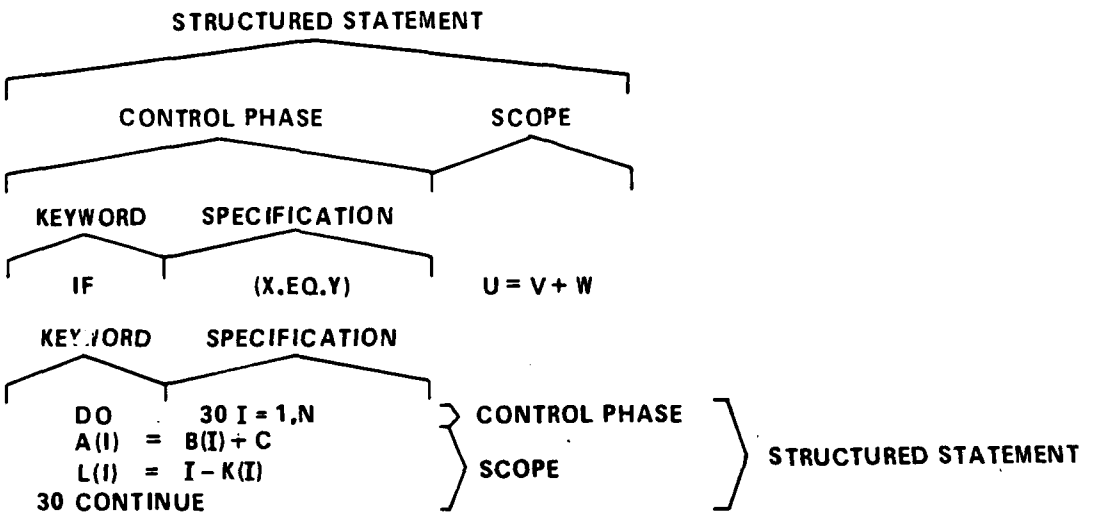
Details of line numbering are machine dependent and are given in paragraph C-10. On most card oriented systems, the line numbers may be placed in columns 76-80 of each card. Other systems may have special provisions for line numbers.

The beginning Flecs programmer should discover and make special note of the details of the mechanism by which Fortran

compiler error messages may be traced back to the Flecs listing on the system being used.

C-4 STRUCTURED STATEMENTS

A basic notion of Flecs is that of the structured statement which consists of a control phrase and its scope. Fortran has two structured statements, the logical IF and the DO. The following examples illustrate this terminology:



Note that each structured statement consists of a control phase which control the execution of a set of one or more statements called its scope. Also note that each control phrase consists of a keyword plus some additional information called the specification. A statement which does not consist of a control phrase and a scope is said to be a simple statement. Examples of simple statements are assignment statements, subroutine CALLs, arithmetic IFs, and GO TOs.

The problem with the Fortran logical IF statement is that its scope may contain only a single simple statement. This restriction is eliminated in the case of the DO, but at the cost of clerical detail (having to stop thinking about the problem while a statement number is invented). Note also that the IF

specification is enclosed in parenthese while the DO specification is not.

In Flecs there is a uniform convention for writing control phrases and indicating their scopes. To write a structured statement, the keyword is placed on a line beginning in column 7 followed by its specification enclosed in parentheses. The remainder of the line is left blank. The statements comprising the scope are placed on successive lines. The end of the scope is indicated by a FIN statement. This creates a multi-line structured statement.

Examples of multi-line structured statements:

```
IF (X.EQ.Y)
   U = V+W
   R = S+T
   FIN

DO (I = 1,N)
   A(I) = B(I)+C
   C = C*2.14-3.14
   FIN
```

Note: The statement number has been eliminated from the DO specification since it is no longer necessary, the end of the loop being specified by FIN.

Nesting of structured statements is permitted to any depth.

Example of nested structured statements:

```
IF (X.EQ.Y)
   U = V+W
   DO (I = 1,N)
      A(I) = B(I)+C
      C = C*2.14-3.14
      FIN
   R = S+T
FIN
```

When the scope of a control phrase consists of a single simple statement, it may be placed on the same line as the control phrase and the FIN may be dispensed with. This creates a one-line structured statement.

Examples of one-line structured statements:

```
IF (X.EQ.Y)  U = V+W
DO (I = 1,N)  A(I) = B(I)+C
```

Since each control phrase must begin on a new line, it is not possible to have a one-line structured statement whose scope consists of a structured statement.

Example of invalid construction:

```
IF (X.EQ.Y)  DO (I = 1,N)  A(I) = B(I)+C
```

To achieve the effect desired above, the IF must be written in a multi-line form.

Example of valid construction:

```
IF (X.EQ.Y)
   DO (I + 1,N) A(I) = B(I)+C
   FIN
```

In addition to IF and DO, Flecs provides several useful structured statements not available in Fortran. After a brief excursion into the subject of indentation, we will present these additional structures.

C-5 INDENTATION, LINES AND THE LISTING

In the examples of multi-line structured statements above, the statements in the scope were indented and an "L" shaped line was drawn connecting the keyword of the control phrase to the matching FIN. The resulting graphic effect helps to reveal the structure of the program. The rules for using indentation and FINs are quite simple and uniform. The control phrase of a multi-line structured statement always causes indentation of the statements that follow. Nothing else causes indentation. A level of indentation (i.e., a scope) is always terminated with a FIN. Nothing else terminates a level of indentation.

When writing a Flecs program on paper, the programmer should adopt the indentation and line drawing conventions shown below. When preparing a Flecs source program in machine readable form, however, each statement should begin in column seven. When the Flecs translator produces the listing, it will reintroduce the correct indentation and produce the corresponding lines. If the programmer attempts to introduce his own

indentation with the use of leading blanks, the program will be translated correctly, but the resulting listing will be improperly indented.

Example of indentation:

1. Program as written on paper by programmer:

```
IF (X.EQ.Y)
   U + V+W
   DO (I = 1,N)
      A(I) = B(I)+C
      C = C*2.14-3.14
      FIN
   R = S+T
   FIN
```

2. Program as entered into computer:

```
IF (X.EQ.Y)
U + V+W
DO (I = 1,N)
A(I) = B(I)+C
C = C*2.14-3.14
FIN
R = S+T
FIN
```

3. Program as listed by Flecs translator:

```
IF (X.EQ.Y)
.  U = V+W
.  DO (I = 1,N)
   .  (A(I) = B(I)+C
   .  C = C*2.14-3.14
.  ...FIN
.  R = S+T
...FIN
```

The correctly indented listing is a tremendous aid in reading and working with programs. Except for the dots and spaces used for indentation, the lines are listed exactly as they appear in the source program. That is, the internal spacing of columns 7-72 is preserved. There is seldom any need to refer to a straight listing of the unindented source.

Comment lines are treated in the following way on the listing to prevent interruption of the dotted lines indicating

scope. A comment line which contains only blanks in columns 2 through 6 will be listed with columns 7 through 72 indented at the then-current level of indentation as if the line were an executable statement. If, however, one or more non-blank characters appear in columns 2 through 6 of a comment card, it will be listed without indentation. Blank lines may be inserted in the source and will be treated as empty comments.

C-6 CONTROL STRUCTURES

The complete set of control structures provided by Flecs is given below together with their corresponding flow charts. The symbol *L* is used to indicate a logical expression. The symbol *S* is used to indicate a scope of one or more statements. Some statements, as indicated below, do not have a one-line construction.

A convenient summary of the information in this chapter may be found in Attachment 1.

C-6.1 DECISION STRUCTURES

Decision structures are structured statements which control the execution of their scopes on the basis of a logical expression or test.

C-6.1.1 IF

Description: The IF statement causes a logical expression to be evaluated. If the value is true, the scope is executed once and control passes to the next statement. If the value is false, control passes directly to the next statement without execution of the scope.

General Form: Flow Chart:

IF (*L*) *S*

Examples:

```
IF (X.EQ.Y) U = V+W

IF (T.GT.0.AND.S.LT.R)
.  I = I+1
.  Z = 0.1
...FIN
```

C-6.1.2 UNLESS

Description: "UNLESS (ℒ)" is functionally equivalent to "IF(.NOT.(ℒ))", but is more convenient is some contexts.

General Form:

```
UNLESS (ℒ) S
```

Flow Chart:

Examples:

```
UNLESS (X.NE.Y) U = V+W

UNLESS (T.LE.O.OR.S.GE.R
.  I = I-1
.  Z = 0.1
...FIN
```

C-6.1.3 WHEN...ELSE

Description: The WHEN...ELSE statements correspond to the IF...THEN...ELSE statement of ALGOL, PL/1, PASCAL, etc. In Flecs, both the WHEN and the ELSE act as structured statements although only the WHEN has a specification. The ELSE statement must immediately follow the scope of the WHEN. The specifier of the WHEN is evaluated and exactly one of the two scopes is executed. The scope of the WHEN statement is executed if the expression is true and the scope of the ELSE statement is executed if the expression is false. In either case, control then passes to the next statement following the ELSE statement.

General Form:

```
WHEN (ℒ) S1
ELSE S2
```

Flow Chart:

Examples:

```
WHEN (X.EQ.Y) U = V+W
ELSE U = V-W

WHEN (X.EQ.Y)
.  U = V+W
.  T = T+1.5
...FIN
ELSE U = V-W

WHEN (X.EQ.Y) U = V+W
ELSE
.  U = V-W
.  T = T+1.5
...FIN
```

```
WHEN (X.EQ.Y)
.  U = V+W
.  T = T-1.5
...FIN
ELSE
.  U = V-W
.  T = T+1.5
...FIN
```

Note: WHEN and ELSE always come as a pair of statements, never separately. Either the WHEN or both may assume the multi-line form. ELSE is considered to be a control phrase, hence cannot be placed on the same line as the WHEN. Thus, "WHEN ($\mathcal{L}$) S_1, ELSE S_2" is <u>not</u> valid.

C-6.1.4 <u>CONDITIONAL</u>

Description: The CONDITIONAL statement is based on the LISP conditional. A list of logical expressions is evaluated one by one until the first expression found to be true is encountered. The scope corresponding to that expression is executed, and control then passes to the first statement following the CONDITIONAL. If all expressions are false, no scope is executed. (See, however, the note about OTHERWISE below.)

General Form:

```
CONDITIONAL
.  (L1)S1
.  (L2)S2
.  .    .
.  .    .
.  .    .
.  (Ln)Sn
...FIN
```

Flow Chart:

Examples:

```
CONDITIONAL
.  (X.LT.-5.0)   U = U+W
.  (X.LE.1.0)    U = U+W+Z
.  (X.LE.10.5)   U = U-Z

CONDITIONAL
.  (A.EQ.B) Z = 1.0
.  (A.LE.C)
.  .  Y = 2.0
.  .  Z = 3.4
.  ...FIN
.  (A.GT.C.AND.A.LT.B) Z = 6.2
.  (OTHERWISE) Z = 0.0
...FIN
```

Notes: The CONDITIONAL itself does not possess a one-line form. However, each "$(\mathcal{L}_i)\ \mathcal{S}_i$ " is treated as a structured statement and may be in one-line or multi-line form.

The reserved word OTHERWISE represents a catchall condition. That is, "(OTHERWISE)$\mathcal{S}_n$" is equivalent to "(.TRUE.)$\mathcal{S}_n$" in a CONDITIONAL statement.

C-6.1.5 SELECT

Description: The SELECT statement is similar to the CONDITIONAL but is more specialized. (It actually is analogous to the PASCAL CASE OF construct.) It allows an expression to be tested for equality to each expression in a list of expressions. When the first matching expression is encountered, a corresponding scope is executed and the SELECT statement terminates. In the description below, $\mathcal{E}, \mathcal{E}, \ldots, \mathcal{E}_n$ represent arbitrary but compatible expressions. Any type of expression (integer, real, complex,...) is allowed as long as the underlying Fortran system allows such expressions to be compared with an .EQ. or .NE. operator.

General Form:

```
SELECT (ℰ)
.  (ℰ)  𝒮
.  (ℰ)  𝒮
.  .     .
.  .     .
.  .     .
.  (ℰn)  𝒮n
...FIN
```

Flow Chart:

Example:

```
SELECT (OPCODE(PC))
.  (JUMP)  PC = AD
.  (ADD)
.  .  A = A+B
.  .  PC = PC+1
.  ...FIN
.  (SKIP)  PC = PC+2
.  (STOP)  CALL STOPCD
...FIN
```

Notes: As in the case of CONDITIONAL, at most one of the S_i will be executed. The catchall OTHERWISE may also be used in a SELECT statement. Thus "(OTHERWISE)S_n" is equivalent to "($\mathcal{E}$)S_n" within a "SELECT ($\mathcal{E}$)" statement.

The expression $\mathcal{E}$ is reevaluated for each comparison in the list, thus lengthy, time consuming, or irreproducible expressions should be precomputed, assigned to a variable, and the variable used in the specification portion of the SELECT statement.

C-6.2 LOOP STRUCTURES

The structured statements described below all have a scope which is executed a variable number of times depending on specified conditions.

Of the five loops presented, the most useful are the DO, WHILE, and REPEAT UNTIL loops. To avoid confusion, the REPEAT WHILE and UNTIL loops should be ignored initially.

C-6.2.1 DO

Description: The Flecs DO loop is functionally identical to the Fortran DO loop. The only differences are syntactic. In the Flecs DO loop, the statement number is omitted from the DO statement, the incrementation parameters are enclosed in parenthesis, and the scope is indicated by either the one line or multi-line convention. Since the semantics of the Fortran DO statement vary from one Fortran compiler to another, a flowchart cannot be given. The symbol I represents any legal incrementation specification.

General Form:

DO (I)S

Equivalent Fortran:

```
    DO 30 I
      S
30  CONTINUE
```

Examples:

```
DO (I = 1,N)  A(I) = 0.0

DO (J =3,K,3)
.  B(J) = B(J-1)*B(J-2)
.  C(J) = SIN(B(J))
...FIN
```

C-6.2.2 WHILE

Description: The WHILE loop causes its scope to be repeatedly executed while a specified condition is true. The condition is checked prior to the first execution of the scope, thus if the condition is initially false the scope will not be executed at all.

General Form: Flow Chart:

WHILE (ℒ) S

Examples:

```
WHILE (X.LT.A(I))  I = I+1
```

```
WHILE (P.NE.O)
.  VAL(P) = VAL(P) +1
.  P = LINK(P)
...FIN
```

C-6.2.3 REPEAT WHILE

Description: By using the REPEAT verb, the test can be logically moved to the end of the loop. The REPEAT WHILE loop causes its scope to be repeatedly executed while a specified condition remains true. The condition is not checked until after the first execution of the scope. Thus the scope will always be executed at least once and the condition indicates under what condition the scope is to be repeated. Note: "REPEAT WHILE (ℒ)" is functionally equivalent to "REPEAT UNTIL(.NOT.(ℒ))".

General Form: Flow Chart:

REPEAT WHILE (ℒ) S

Examples:

```
REPEAT WHILE(N.EQ.M(I))  I = I+1
```

```
REPEAT WHILE (LINK(Q).NE.0)
.  R = LINK(Q)
.  LINK(Q) = P
.  P = Q
.  Q = R
...FIN
```

C-6.2.4 UNTIL

Description: The UNTIL loop causes its scope to be repeatedly executed until a specific condition becomes true. The condition is checked prior to the first execution of the scope, thus if the condition is initially true, the scope will not be executed at all. Note that "UNTIL (*L*)" is functionally equivalent to "WHILE (.NOT.(*L*))".

General Form: Flow Chart:

```
UNTIL (L) S
```

Examples:

```
UNTIL (X.EQ.A(I))  I = I+1
```

```
UNTIL (P.EQ.0)
.  VAL(P) = VAL(P) +1
.  P = LINK(P)
...FIN
```

C-6.2.5 REPEAT UNTIL

Description: By using the REPEAT verb, the test can be logically moved to the end of the loop. The REPEAT UNTIL loop causes its scope to be repeatedly executed until a specified condition becomes true. The condition is not checked until after the first execution of the scope. Thus, the scope will always be executed at least once and the condition indicates under what conditions the repetition of the scope is to be terminated.

General Form: Flow Chart:

```
REPEAT UNTIL (L) S
```

Examples:

```
REPEAT UNTIL (N.EQ.M(I))  I = I+1
```

```
REPEAT UNTIL (LINK(Q).EQ.0)
.  R = LINK(Q)
.  LINK(Q) = P
.  P = Q
.  Q = R
...FIN
```

C-7 INTERNAL PROCEDURES

In Flecs a sequence of statements may be declared an internal procedure and a given name. The procedure may then be invoked from any point in the program by simply giving its name.

Procedure names may be any string of letters, digits, and hyphens (i.e., minus signs) beginning with a letter and containing at least one hyphen. Internal blanks are not allowed. The only restriction on the length of a name is that it may not be continued onto a second line.

Examples of valid internal procedure names:

```
INITIALIZE-ARRAYS
GIVE-WARNING
SORT-INTO-DESCENDING-ORDER
INITIATE-PHASE-3
```

A procedure declaration consists of the keyword "TO" followed by the procedure name and its scope. The set of statements comprising the procedure is called its scope. If the scope consists of a single simple statement it may be placed on the same line as the "TO" and procedure name, otherwise the statements of the scope are placed on the following lines and terminated with a FIN statement. These rules are analogous with the rules for forming the scope of a structured statement.

General Form of procedure declaration:

```
TO procedure-name
```

Examples of procedure declarations:

```
TO RESET-POINTER P = 0

TO DO-NOTHING CONTINUE

TO SUMMARIZE-FILE
.  INITIALIZE-SUMMARY
.  OPEN-FILE
.  REPEAT UNTIL (EOF)
.  .  ATTEMPT-TO-READ-RECORD
.  .  WHEN (EOF) CLOSE-FILE
.  .  ELSE UPDATE-SUMMARY
.  ...FIN
.  OUTPUT-SUMMARY
...FIN
```

An internal procedure reference is a procedure name appearing where an executable statement would be expected. In fact an internal procedure reference is an executable simple statement and thus may be used in the scope of a structured statement as in the last example above. When control reaches a procedure reference during execution of a Flecs program, a return address is saved and control is transferred to the first statement in the scope of the procedure. When control reaches the end of the scope, control is transferred back to the statement logically following the procedure reference.

A typical Flecs program or subprogram consists of a sequence of Fortran declarations: (e.g., INTEGER, DIMENSION, COMMON, etc.) followed by a sequence of executable statements called the body of the program followed by the Flecs internal procedure declarations, if any, and finally the END statement.

Here is a Flecs program which illustrates the placement of the procedure declarations.

```
00010  C  INTERACTIVE PROGRAM FOR PDP-10 TO COMPUTE X**2.
00020  C  ZERO IS USED AS A SENTINEL VALUE TO TERMINATE EXECUTION.
00030
00040          REAL X,XSQ
00050          REPEAT UNTIL (X.EQ.0)
00060          .  GET-A-VALUE-OF-X
00070          .  IF (X.NE.0)
00080          .  .  COMPUTE-RESULT
00090          .  .  TYPE-RESULT
00100          .  ...FIN
00110          ...FIN
00120          CALL EXIT
  ------------------------------------------------------
00130          TO-GET-A-VALUE-OF-X
00140          .  TYPE 10
00150  10      .  FORMAT (' X = ' ,$)
00160          .  ACCEPT 20,X
00170  20      .  FORMAT (F)
00180          ...FIN
  ------------------------------------------------------
00190          TO COMPUTE-RESULT XSQ = X*X
  ------------------------------------------------------
00200          TO TYPE-RESULT
00210          .  TYPE 30,XSQ
00220  30      .  FORMAT(' X-SQUARED - ',F7.2)
00230          ...FIN
00240          END
```

Notes concerning internal procedures:

1. All internal procedure declarations must be placed at the end of the program just prior to the END statement. The appearance of the first "TO" statement terminates the body of the program. The translator expects to see nothing but procedure declarations from that point on.
2. The order of the declarations is not important. Alphabetical by name is an excellent order for programs with a large number of procedures.
3. Procedure declarations may not be nested. In other words, the scope of a procedure may not contain a procedure declaration. It may, of course, contain executable procedure references.
4. Any procedure may contain references to any other procedures (excluding itself).
5. Dynamic recursion of procedure referencing is not permitted.
6. All program variables within a main or subprogram are global and are accessible to the statements in all procedures declared within that same main or sub program.
7. There is no formal mechanism for defining or passing parameters to an internal procedure. When parameter passing is needed, the Fortran function or subroutine subprogram mechanism may be used or the progammer may invent his own parameter passing methods using the global nature of variables over internal procedures.
9. The Flecs translator separates procedure declarations on the listing by dashed lines as shown in the preceding example.

C-8 RESTRICTIONS AND NOTES

If Flecs were implemented by a compiler this section would be much shorter. Currently, however, Flecs is implemented

by a sturdy but naive translator. Thus, the Flecs programmer must observe the following restrictions:

1. Flecs must invent many statement numbers in creating the Fortran program. It does so by beginning with a large number (in this implementation 32767) and generating successively smaller numbers as it needs them. Do not use a number which will be generated by the translator. A good rule of thumb is to avoid using 5 digit statement numbers.
2. The Flecs translator must generate integer variable names. It does so by using names of the form "Innnnn" when nnnnn is a five digit number related to a generated statement number. Do not use variables of the form Innnnn and avoid causing them to be declared other than INTEGER. For example, the declaration "IMPLICIT REAL (A-Z)" leads to trouble. Try "IMPLICIT REAL (A-H, J-Z)" instead.
3. The translator does not recognize continuation lines in the source file. Thus Fortran statements may be continued since the statement and its continuations will be passed through the translator without alteration. However, an extended Flecs statement which requires translation may not be continued. The reasons one might wish to continue a Flecs statement 1) It is a structured statement or procedure declaration with a one statement scope too long to fit on a line, or 2) it contains an excessively long specification portion, or 3) both of the above. Problem 1) can be avoided by going to the multi-line form. Frequently problem 2) can be avoided when the specification is an expression (logical or otherwise) by assigning the expression to a variable in a preceding statement and then using the variable as the specification.

4. Blanks are meaningful separators in Flecs statements; don't put them in dumb places like the middle of identifiers or key words and do use them to separate distinct words like REPEAT and UNTIL.

5. Let Flecs indent the listing. Start all statements in Col. 7 (or use the TAB key) and the listing will always reveal the true structure of the program. (As understood by the translator, of course.)

6. As far as the translator is concerned, FORMAT statements are executable Fortran statements since it doesn't recognize them as extended Flecs statements. Thus, only place FORMAT statements where an executable Fortran statements would be acceptable. Don't put them between the end of a WHEN statement and the beginning of an ELSE statement. Don't put them between procedure declarations.

Incorrect Examples:

```
     WHEN (FLAG) WRITE(3,30)
 30  FORMAT(7H TITLE:)
     ELSE LINE = LINE+1


     TO WRITE-HEADER
     .  PAGE = PAGE +1
     .  WRITE(3,40) H,PAGE
     ...FIN
 40  FORMAT (70A1,I3)
```

Corrected Examples:

```
     .WHEN (FLAG)
 30  .  WRITE(3,30)
     .  FORMAT(7H TITLE:)
     ...FIN
     ELSE  LINE = LINE+1

     TO WRITE-HEADER
     .  PAGE = PAGE+1
     .  WRITE(3,40) H,PAGE
 40  .  FORMAT(70A1,I3)
     ...FIN
```

7. The translator recognizes extended Flecs statements by the process of scanning the first identifier on the line. If the identifier is one of the Flecs keywords, IF, WHEN, UNLESS, FIN, etc., the line is assumed to be a Flecs statement and is treated as such. Thus, the Flecs keywords are reserved and may not be used as variable names. In case of necessity, a variable name, say WHEN, may be slipped past the translator by embedding a blank within it. Thus "WH EN" will look like "WH" followed by "EN" to the translator which is blank

sensitive, but like "WHEN" to the compiler which ignores blanks.

8. In scanning a parenthesized specification, the translator scans from left to right to find the parenthesis which matches the initial left parenthesis of the specification. The translator, however, is ignorant of Fortran syntax including the concept of Hollerith constants and will treat Hollerith parenthesis as syntactic parenthesis. Thus, avoid placing Hollerith constants containing unbalanced parenthesis within specifications. If necessary, assign such constants to a variable, using a DATA or assignment statement, and place the variable in the specification.

Incorrect Example:	Corrected Example:
IF(J.EQ.'(')	LP = '(' IF(J.EQ.LP)

9. The Flecs translator will not supply statements necessary to cause appropriate termination of main and sub-programs. Thus, it is necessary to include the appropriate RETURN, STOP, or CALL EXIT statement prior to the first internal procedure declaration. Failure to do so will result in control entering the scope of the first procedure after leaving the body of the program. Do not place such statements between the procedure declarations and the END statement.

C-9 ERRORS

This section provides a frameword for understanding the error handling mechanisms of version 22 of the Flecs Translator. The system described below is at an early point in evolution, but has proven to be quite workable.

The Flecs translator examines a Flecs program on a line by line basis. As each line is encountered it is first subjected to a limited syntax analysis followed by a context

analysis. Errors may be detected during either of these analyses. It is also possible for errors to go undetected by the translator.

C-9.1 Syntax Errors

When a syntax error is detected by the translator, it ignores the statement. On the Flecs listing the line number of the statement is overprinted with -'s to indicate that the statement has been ignored. The nature of the syntax error is given in a message on the following line.

The fact that a statement has been ignored may, of course, cause some context errors in later statements. For example, the control phrase "WHEN (X(I).LT.(3+4)" has a missing right parenthesis. This statement will be ignored, causing as a minimum the following ELSE to be out of context. The programmer should of course be aware of such effects. More is said about them in the next section.

C-9.2 Context Errors

If a statement successfully passes the syntax analysis, it is checked to see if it is in the appropriate context within the program. For example, an ELSE must appear following a WHEN and nowhere else. If an ELSE does not appear at the appropriate point of if it appears at some other point, a context error has occurred. A frequent source of context errors in the initial stages of development of a program comes from miscounting the number of FINs needed at some point in the program.

With the exception of excess FINs which do not match any preceding control phrase and are ignored (as indicated by overprinting the line number). all context errors are treated with a uniform strategy. When an out-of-context source statement is encountered, the translator generates a "STATEMENT(S) NEEDED" message. It then invents and processes a sequence of statements which, if they had been included at that point in the program, would have placed the original source statement in a correct context. A message is given for each such statement

invented. The original source statement is then processed in the newly created context.

By inventing statements the translator is not trying to patch up the program so that it will run correctly, it is simply trying to adjust the local context so that the original source statement and the statements which follow will be acceptable on a context basis. As in the case of context errors generated by ignoring a syntactically incorrect statement, such an adjustment of context frequently causes further context errors later on. This is called propagation of context errors.

One nice feature of the context adjustment strategy is that context errors cannot propagate past a recognizable procedure declaration. This is because the "TO" declaration is in context only at indentation level 0. Thus to place it in context, the translator must invent enough statements to terminate all open control structures which preceed the "TO". The programmer who modularizes his program into a collection of relatively short internal procedures, limits the potential for propagation of context errors.

C-9.3 Undetected Errors

The Flecs translator is ignorant of most details of Fortran syntax. Thus most Fortran syntax errors will be detected by the Fortran compiler not the Flecs translator. In addition there are two major classes of Flecs errors which will be caught by the compiler not the translator.

The first class of undetected errors involve misspelled Flecs keywords. A misspelled keyword will not be recognized by the translator. The line on which is occurs will be assumed to be a Fortran statement and will be passed unaltered to the compiler which will no doubt object to it. For example a common error is to spell UNTIL with two Ls. Such statements are passed to the compiler, which then produces an error message. The fact that an intended control phrase was not recognized frequently causes a later context error since a level of indentation will not be triggered.

The second class of undetected errors involves unbalanced parentheses. When scanning a parenthesized specification, the translator is looking for a matching right parenthesis. If the matching parenthesis is encountered before the end of the line the remainder of the line is scanned. If the remainder is blank or consists of a recognizable internal procedure reference, all if well. If neither of the above two cases hold, the remainder of the line is assumed (without checking) to be a simple Fortran statement which is passed to the compiler. Of course, this assumption may be wrong, thus the statement

```
"WHEN (X.LT.A(I)+Z)) X = 0"
```

is broken into

```
keyword "WHEN"
specification "(X.LT.A(I)+Z)"
Fortran statement ") X = 0"
```

Needless to say the compiler will object to ") X = 0" as a statement.

Programmers on batch oriented systems have less difficulty with undetected errors due to the practice of running the program through both the translator and the compiler each time a run is submitted. The compiler errors usually point out any errors undetected by the translator.

Programmers on timesharing systems tend to have a bit more difficulty since an undetected error in one line may trigger a context error in a much later line. Noticing the context error, the programmer does not proceed with compilation and hence is not warned by the compiler of the genuine cause of the error. One indication of the true source of the error may be an indentation failure at the corresponding point in the listing.

C-9.4 Other Errors

The translator detects a variety of other errors such as multiply defined, or undefined procedure references. The error message are self-explanatory. (Really and truly!)

C-10 PROCEDURES FOR USE ON THE PDP-11/34

C-10.1 Source Preparation

Prepare a Flecs source file with a name of your choosing and an extension of ".FLX".

C-10.2 Translator Commands

Flecs is installed under the task name "...FLE".

Flecs may be invoked by the following command string: (Items in brackets ([]) are optional. <CR> denotes carriage return.)

FLE [OUTPUT] [,LIST] = INPUT <CR>

where: [OUTPUT] is the output filename. This file will be given an extension of .FTN, and may be input directly to the Fortran compiler:

[,LIST] is the listing device or listing filename. If a device is specified it must be a list device such as LP: or TI:. If a filename is specified it will be given an extension ".FLL";

INPUT is the input filename. This should contain the FLECS source statements and must have an extension of ".FLX".

If OUTPUT or LIST is omitted, no file will be created. If both OUTPUT and LIST are omitted, a list of file with name RSXFLEX.FLL and the next highest version will be created.

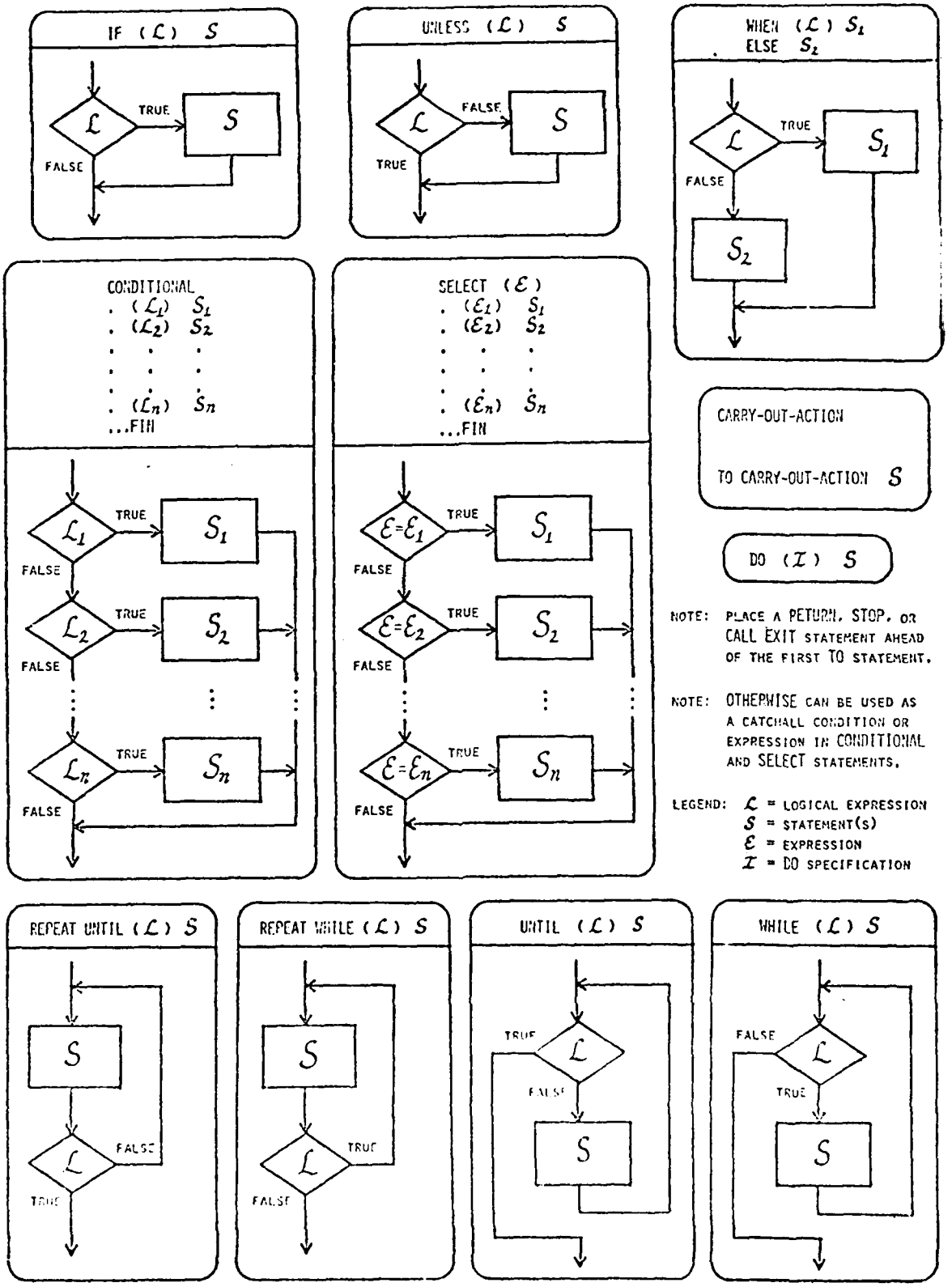
Example: a source file named 'EXAMPLE.FLX' has been created. In order to translate the Flecs program into a Fortran program, type the command stream

FLE EXAMPLE,EXAMPLE = EXAMPLE<CR>

A Fortran source file with the name EXAMPLE.FTN and a Flecs listing file with the name EXAMPLE.FLL will be created.

ATTACHMENT 1

Flecs Summary Sheet



APPENDIX D

DEFINITION OF PUBLIC VARIABLES

```
ASAVE :   BYTE RESERVED FOR SAVING THE A REGISTER
ASCBUF:   FIVE CHARACTER ASCII BUFFER FOR DVCNVT ROUTINE
BSAVE :   BYTE RESERVED FOR SAVING THE B REGISTER
CSAVE :   BYTE RESERVED FOR SAVING THE C REGISTER
CURNTF:   5 BYTE ARRAY CONTAINING ASCII REPRESENTATION
CYCDON:   FLAG=0FFH IF A CYCLE TEST IS DONE.
CYCNBR:   NUMBER OF THE CURRENT CYCLE OF A MULTICYCLE TEST
CYCNMR:   MODE OF OPERATION FLAG=0=SINGLE-1,2,3=MULTI CYCLE
DVM1  :   DVM1 DATA
DVM2  :   DVM2 DATA
DCD   :   DECADE INCREMENTER
DECCNT:   DECADE DOWN COUNTER FOR LOOP (GENERQ)
DECNUM:   CURRENT DECADE NUMBER (1 THRU 6)
DECSW :   DECADE SWITCH BUFFER (6 BYTES)
DELDON:   FLAG=0FFH IF PREVIOUS FREQUENCY WAS DELETED
DSAVE :   BYTE RESERVED FOR SAVING THE D REGISTER
DVM1ON:   DVM1 ON FLAG (ON=1)
DVM2ON:   DVM2 ON FLAG (ON=1)
DX    :   X AXIS STEP SIZE FOR CURRENT DECADE
E6BYTE:   PORT E6 COMMAND BUFFER
EABYTE:   PORT EA COMMAND BUFFER
ESAVE :   E REGISTER SAVE LOCATION
EXPAND:   EXPANDED GRID FLAG (NO EXPAND=0, EXPAND=1)
EXPIRD:   INTER CYCLE INTERVAL FLAG=0FFH WHEN INTERVAL HAS EXPIRD
F4H   :   PORT F4 BUFFER DVM1 LS
F5H   :   PORT F5 BUFFER DVM1 MIDDLE
F6BYTE:   PORT F6 COMMAND BUFFER
F6H   :   PORT F6 BUFFER DVM1 MS
F8H   :   PORT F8 BUFFER DVM2 LS
F9H   :   PORT F9 BUFFER DVM2 MIDDLE
FABYTE:   PORT FA COMMAND BUFFER
FAH   :   PORT FA BUFFER DVM2 MS
FIFBUF:   4000 BYTE BUFFER FOR THE RF MAGNITUDE DATA
FIFCNT:   FIFBUF BYTE COUNT
FIFIN :   FIFBUF INPUT POINTER
FIFOUT:   FIFBUF OUTPUT POINTER
FORMAT:   FORMAT SWITCH STATUS
FRQBUF:   12000 BYTE BUFFER FOR THE RF PHASE DATA
FRQPTR:   FRQBUF POINTER
FSAVE :   BYTE RESERVED  TO SAVE THE F REGISTER
GRDON :   FLAG=1 TO GENERATE GRID. USED BY CYCCHK MODULE
GRIDON:   GRID GENERATION & DISPLAY (0=NO , 1=YES)
HLDFLG:   CONTINUE CYCLE=0, HOLD CYCLE=0FFH
HSAVE :   BYTE RESERVED TO SAVE THE H REGISTER
INTER0:   INTERRUPT 0 SERVICE ROUTINE VECTOR
INTER1:    "          1
INTER2:    "          2
INTER3:    "          3
INTER4:    "          4
INTER5:    "          5
INTER6:    "          6
INTER7:    "          7
```

```
INTCTR:  BINARY COUNTER FOR SYSTEM STEP PULSES
INTLN1:  BINARY VALUE EQUAL TO THE NUMBER OF SYSTEM
         STEP PULSES FOR DELAY BETWEEN CYCLE 1 AND 2
INTLN2:  SAME FOR CYCLE2/3
JTBL   : START ADDRESS FOR THE INTERRUPT JUMP TABLE
LAST   : LAST FORMAT PLOTTED WITH GRIDS ON
LOGPTR:  LOG TABLE POINTER
LSAVE  : BYTE RESERVED TO SAVE THE L REGISTER
MFE    : FLAG=1 INDICATES MFE CASSETTE IS ON LINE
MIDSTP:  MID PROGRAM STOP FLAG (MPSTOP = 1)
MLTCYC:  SINGLE CYCLE TEST=0, MULTICYCLE TEST=1
MODE   : MODE SWITCH STATUS
MRAMPT:  MULTIPLE DELETE RAM TABLE POINTER
MROMPT:  MULTIPLE DELETE ROM TABLE POINTER
MSTACK:  TOP OF THE 1024 BYTE STACK
NACT   : NUMBER OF ACTIVE DECADES
NLOAD  : NUMBER OF ASCII CHARACTERS TO LOAD IN SD1FRQ
NPTS   : NO. OF POINTS/DEC IN CURRENT DECADE
NXSEMI:  FLAG=OFFH IF NEXT STEP OF A SEMI-AUTOMATIC TEST IS TO BE DONE
NXTCYC:  FLAG=OFFH IF MORE CYCLES TO DO IN MULTICYCLE
PASSNO:  PASS THRU CWPLOT ROUTINE 0=X DATA 1=YDATA
PDP11  : FLAG=1 INDICATES PDP11 IS ON LINE
PENFLG:  PEN COMMAND FLAG
PHFLAG:  PHASE TEST FLAG (PHASE TEST REQ'D = 1)
PINON  : PLOT/INITIATE SWITCH INTERRUPT ENABLE (ENAB=1)
PLOTON:  PLOTTER ON FLAG (ON=1)
PSAVE  : PROGRAM COUNTER SAVE REGISTERS
PSTP   : PROGRAM STOP FLAG (PSTP = 1 , PSTOP = 0)
RAMPTR:  MULTIPLE DELETE RAM TABLE HEAD
RCVRON:  RECEIVER STATUS FLAG ( OFF = 0 , ON = 1 )
REFON  : REFERENCE SYNTHESIZER ON FLAG (ON=1)
RFAMPL:  5 BYTE ARRAY CONTAINING ASCII REPRESENTATION
         OF CURRENT RF AMPLITUDE AND SIGN FROM DVM.
RFPHAS:  5 BYTE ARRAY CONTAINING ASCII REPRESENTATION
         OF CURRENT RF PHASE AND SIGN FROM DVM.
ROMPTR:  MULTIPLE DELETE ROM TABLE HEAD
RSTART:  FLAG=OFFH IF SYSTEM STEP PULSES NEED TO
         BE RE-INITIATED BY SYNCHRONIZER CLOCK
SCAL   : ROM SCALE TABLE POINTER FOR LOGSCL COPY
SD1PTR:  SD1FRQ BUFFER POINTER
SD2PTR:  SD2FRQ BUFFER POINTER
SERON  : INDICATES SERIAL IO ON
SIGN   : POLARITY SIGN IN DVCNVT
SIZBIN:  BINARY VALUE OF THE SYSTEM STEP TIME INTERVAL
SKIPCT:  SKIP DOWN COUNTER FOR LOGTAB FREQ SKIPS
SKIPSZ:  NO. OF LOGTAB FREQ'S TO  BE SKIPPED
SMPLS  : SAMPLES (OVERLAYS IN SCAL BUFFER)
SRAMPT:  SINGLE DELETE RAM TABLE POINTER
SROMPT:  SINGLE DELETE ROM TABLE POINTER
SSAVE  : STACK POINTER SAVE REGISTER
STATE  : TEST STATE
STEPCT:  FREQUENCY STEP DOWN COUNTER (NPTS -> 0)
SYPASS   1=1ST PASS, 0=LATER PASSES
TAPEON:  TAPE CASETTE ON FLAG (ON=1)
TEMP   : MAIN TEMPORARY  BUFFERS
TEMP1  : TEMPORARY MONITOR CELL
TIMER4:  SET FLAG TO OFFH TO ACTIVATE INTER-MESSAGE
```

	DELAY [illegible] TO PLOT [illegible]
LINEDS	SET FLAG TO START ACTIVE DELAY BETWEEN CONSECUTIVE Z180 COMMANDS.
[illegible]	ASCII TEST NAME
MSPTAC	SET TO OPEN WHEN THE DATA LINK IS BUSY
VTOON	VTO SYNTHESIZER ON FLAG (ON=1)
XI	X COORDINATE LOCATION
XVAL	X AXIS PLOTTER POSITION ACCUMULATOR
YI	Y COORDINATE DATA (OVERLAYS IN XI BUFFER)
YOFSET	Y AXIS OFFSET FOR SELECTED GRID FORMAT(CONFIG)
ZINT	Z180 INTERRUPT FLAG (SET BY INTERRUPT ZTINT)

APPENDIX E

SUBMIT FILE FOR LINK/LOCATE WITH CODE IN ROM

```
LINK &
 F1:RESET.OBJ, &
 F1:PARIO.OBJ, &
 F1:TIMEO.OBJ, &
 F1:INTIO.OBJ, &
 F1:ZTIOO.OBJ, &
 F1:WHODAT.OBJ, &
 F1:ZNOC.OBJ, &
 F1:CKDVMI.OBJ, &
 F1:PINIT.OBJ, &
 F1:SWPOLE.OBJ, &
 F1:CONFIG.OBJ, &
 F1:GENERO.OBJ, &
 F1:DECDAT.OBJ, &
 F1:DELPT.OBJ, &
 F1:STEP.OBJ, &
 F1:SDIRQ.OBJ, &
 F1:DELID.OBJ, &
 F1:CWTEST.OBJ, &
 F1:CWPLOT.OBJ, &
 F1:CKMIDS.OBJ, &
 F1:LODPLT.OBJ, &
 F1:SYSTEP.OBJ, &
 F1:DVNPOL.OBJ, &
 F1:DVCNVT.OBJ, &
 F1:XPOSN.OBJ, &
 F1:SDDEC.OBJ, &
 F1:DVIINT.OBJ, &
 F1:START.OBJ, &
 F1:ZTINT.OBJ, &
 F1:PARIOS.OBJ, &
 F1:ERRSG.OBJ, &
 F1:ZTIO.OBJ, &
 F1:PLUTO.OBJ, &
 F1:LABAXS.OBJ, &
 F1:YLBL.OBJ, &
 F1:NONEN.OBJ, &
 F1:STAND.OBJ, &
 F1:STNANT.OBJ, &
 F1:REFLV.OBJ, &
 F1:ROTATE.OBJ, &
 F1:INCK.OBJ, &
 F1:MVCTR.OBJ, &
 F1:STRTEN.OBJ, &
 F1:XINITS.OBJ, &
 F1:XLABEL.OBJ, &
 F1:SAMPLE.OBJ, &
 F1:ZTGO.OBJ, &
 F1:GENGRO.OBJ, &
 F1:DATAI.OBJ, &
 F1:DECADE.OBJ, &
 F1:LOGSCL.OBJ, &
```

```
:F1:PLOTS.OBJ, &
:F1:NXDEC.OBJ, &
:F1:YLNS.OBJ, &
   :F1:BCTR.OBJ, &
   :F1:PACK.OBJ, &
:F1:UNPACK.OBJ, &
:F1:BUF1.OBJ, &
:F1:BCDA.OBJ, &
:F1:BNASC.OBJ, &
:F1:MATH.OBJ, &
:F1:FDUMP.OBJ, &
:F1:FWRAP.OBJ, &
:F1:MOVE.OBJ, &
:F1:DELAY1.OBJ, &
:F1:FLOAD.OBJ, &
:F1:RESTOR.OBJ, &
:F1:DELAY.OBJ, &
:F1:FRET.OBJ, &
:F1:SRET.OBJ, &
:F1:RAM80.OBJ, &
:F1:CWAIT.OBJ, &
:F1:ZTMSK.OBJ, &
:F1:PAT.OBJ, &
:F1:STOP.OBJ, &
PUBLICS(:F1:Z180.ROM), &
PUBLICS(:F1:SDANIB.ROM), &
:F1:TIMER.OBJ, &
:F1:SDINIT.OBJ, &
:F1:SDANPL.OBJ, &
:F1:DBMDBW.OBJ, &
:F1:SDXNTR.OBJ, &
:F1:MODSET.OBJ, &
:F1:CYCTIM.OBJ, &
:F1:CNVINT.OBJ, &
:F1:PNLBLK.OBJ, &
:F1:STABLK.OBJ, &
:F1:MSRBLK.OBJ, &
:F1:REFRQ.OBJ, &
:F1:DELBLK.OBJ, &
:F1:DVMASC.OBJ, &
:F1:CYCCHK.OBJ, &
:F1:HLDTST.OBJ, &
:F1:CYCINV.OBJ, &
:F1:BINASC.OBJ, &
:F1:ASCBIN.OBJ, &
:F1:BCDASC.OBJ, &
:F1:BCDBNY.OBJ, &
:F1:SERLIO.OBJ, &
:F1:RAM116.OBJ, &
:F0:PLM80.LIB &
TO &
:F1:CWMS.LNK

LOCATE &
:F1:CWMS.LNK &
TO &
```

```
F1 CWNS.RUN &
MAP PRINT( F1.CWRON.MAP) PUBLICS CODE(0000H) DATA(3000H) STACKS(ZH-(0)
```

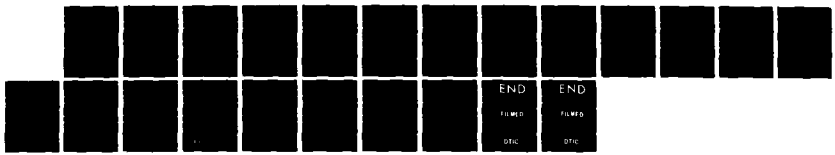
APPENDIX F

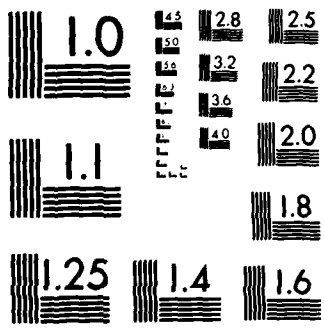
SUBMIT FILE FOR EMULATION WITH CODE IN ROM

```
1CF00
BASE HEX
XF IO 0DH U
XF IO 0EH U
XF IO 0FH U
XF MEM 0 U
XF MEM 1 U
XF MEM 2 U
XF MEM 3 U
XF MEM 4 U
XF MEM 5 U
XF MEM 6 U
XF MEM 7 U
XF MEM 0DH U
XF MEM 0EH U
XF MEM 0FH U
LOAD F1 Z180.ROM
LOAD F1 SDANTIB.ROM
LOAD F1:CWMS.ROM
```

AD-A151 622 CW MEASUREMENT SYSTEM SOFTWARE SYSTEM MAINTENANCE MANUAL(U) EG AND G WASHINGTON ANALYTICAL SERVICES CENTER INC ALBUQUERQU... R NELSON ET AL. 02 APR 82

UNCLASSIFIED EG/G-AG-1435 DNA-6232F DNA001-80-C-0290 NL





APPENDIX G

SUBMIT FILE FOR LINK/LOCATE WITH CODE IN RAM

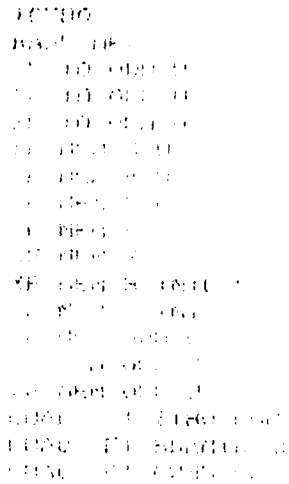
```
LINK &
:F1:RESET.OBJ, &
:F1:PARIO.OBJ, &
:F1:TIMEO.OBJ, &
:F1:INTIO.OBJ, &
:F1:ZTIOO.OBJ, &
:F1:WHODAT.OBJ, &
:F1:ZNOC.OBJ, &
:F1:CKDVM1.OBJ, &
:F1:PINIT.OBJ, &
:F1:SWPOLE.OBJ, &
:F1:CONFIG.OBJ, &
:F1:GENFRQ.OBJ, &
:F1:DECDAT.OBJ, &
:F1:DELPT.OBJ, &
:F1:STEP.OBJ, &
:F1:SDFRQ.OBJ, &
:F1:DELID.OBJ, &
:F1:CWTEST.OBJ, &
:F1:CWPLOT.OBJ, &
:F1:CKMIDS.OBJ, &
:F1:LODPLT.OBJ, &
:F1:SYSTEP.OBJ, &
:F1:DVNPOL.OBJ, &
:F1:DVCNVT.OBJ, &
:F1:XPOSN.OBJ, &
:F1:SDDEC.OBJ, &
:F1:DV1INT.OBJ, &
:F1:START.OBJ, &
:F1:ZTINT.OBJ, &
:F1:PARIOS.OBJ, &
:F1:ERMSG.OBJ, &
:F1:ZTIO.OBJ, &
:F1:PLOTO.OBJ, &
:F1:LABAXS.OBJ, &
:F1:YLBL.OBJ, &
:F1:NOMEN.OBJ, &
:F1:STAND.OBJ, &
:F1:STNANT.OBJ, &
:F1:REFLV.OBJ, &
:F1:ROTATE.OBJ, &
:F1:INCK.OBJ, &
:F1:MVCTR.OBJ, &
:F1:STRTEN.OBJ, &
:F1:XINITS.OBJ, &
:F1:XLABEL.OBJ, &
:F1:SAMPLE.OBJ, &
:F1:ZT80.OBJ, &
:F1:GENSRD.OBJ, &
:F1:DATA1.OBJ, &
:F1:DECADE.OBJ, *
:F1:LOGSCL.OBJ, &
```

```
:F1:PLOTS.OBJ, &
:F1:NXDEC.OBJ, &
:F1:YLNS.OBJ, &
:F1:RSTR.OBJ, &
 F1:PACK.OBJ, &
:F1:UNPACK.OBJ &
 F1:SUB1.OBJ &
:F1:BCDBIN.OBJ &
 F1:BNASC.OBJ, &
:F1:MATH.OBJ, &
 F1:FDUMP.OBJ, &
:F1:FWRAP.OBJ, &
:F1:MOVE.OBJ, &
:F1:DELAY1.OBJ, &
:F1:FLOAD.OBJ, &
:F1:RESTOR.OBJ, &
:F1:DELAY.OBJ, &
:F1:FRET.OBJ, &
:F1:SRET.OBJ, &
:F1:RANBO.OBJ, &
:F1:CWAIT.OBJ, &
:F1:ZTMSK.OBJ, &
:F1:PAT.OBJ, &
:F1:STOP.OBJ, &
PUBLICS(:F1:Z180.ROM), &
PUBLICS(:F1:SDAM1B.ROM), &
:F1:TIMER.OBJ, &
:F1:SDINIT.OBJ, &
:F1:SDAMPL.OBJ, &
:F1:DBMDBW.OBJ, &
 F1:SDXNTR.OBJ, &
:F1:MODSET.OBJ, &
:F1:CYCTIM.OBJ, &
:F1:CNVINT.OBJ, &
:F1:PNLBLK.OBJ, &
:F1:STABLK.OBJ, &
:F1:MSRBLK.OBJ, &
:F1:REFRQ.OBJ, &
:F1:DELBLK.OBJ, &
:F1:DVMASC.OBJ, &
:F1:CYCCHK.OBJ, &
:F1:HLDTST.OBJ, &
 F1:CYCINV.OBJ, &
:F1:BCDASC.OBJ, &
:F1:ASCBIN.OBJ, &
:F1:BCDASC.OBJ, &
 F1:BCDBNY.OBJ, &
 F1:SCPLIO.OBJ, &
 F1:RAN116.OBJ &
:F0:PLM80.LIB &
TO &
:F1:CWMS.LNK

LOCATE &
:F1:CWMS.LNK &
TO &
```

APPENDIX H

SUBMIT FILE FOR EMULATION WITH CODE IN RAM



APPENDIX I

VALIDATION OF THE FORWARD AND INVERSE FOURIER TRANSFORMS USED BY THE CW MEASUREMENT SYSTEM

The purpose of this collection of plots is to demonstrate the correctness of the forward and inverse Fourier transforms used in this software package. Figure I-1 shows a time plot of the equation

$$f(t) = 1.0E5\ (e^{-2E6t} - e^{-4E6t})$$

and the frequency curve generated when this function is processed by the off line forward Fourier transform module.

Integrated directly by hand f(t) has a Fourier transform F(w) with a magnitude equal to:

$$5E10\ (1/(4E24 + 5E12*\pi^2*\omega^2 + \pi^4\omega^4))^{1/2}$$

and a phase equal to:

$$\arctan\ ((-3E6*\pi*\omega)/(2E12 - \pi^2*\omega^2)).$$

$F(\omega)$ is plotted in Figure I-2. Figure I-3 shows the hand integrated and the machine calculated transforms plotted on the same graph. They match so completely the two lines appear as one, demonstrating that the off line forward transform produces the correct results.

Figure I-4 shows the original curve f(t) and the results of processing f(t) through the forward and then the inverse off line transforms plotted on the same graph. Again the lines overlay almost exactly, verifying the inverse off line

transform. Figure I-5 shows the same test performed on a damped sine wave.

Figure I-6 is a typical temps wave form. Figure I-7 is the same data after processing by the off line forward and inverse transforms. Figure I-8 is an overlay plot of the original and transformed threatwave.

Figure I-9 shows frequency data collected from an on line test using a 15 mega hertz filter and the resulting time plot when a pure Butterworth threat was applied to the data and then the on line inverse Fourier transform performed. For Figure I-10 the stored frequency data from this test was multiplied by the Butterworth filter and then processed by the off line inverse transform. Figure I-11 shows the results of the on line and off line inverse transforms plotted on the same graph. Again the two curves are identical, demonstrating that the on line and off line inverse transforms produce the same results.

The plots in this package show that the off line forward Fourier transform is accurate, that transforming a function forward and then back reproduces the original function, and that the on line and off line functions produce the same results. This verifies the correctness of all three routines.

10:22:58 28-JUN-82		MINIMUM FREQUENCY	1.000000E+03	SIGNAL PROBE ID	
TEST SEQUENCE #	1234	MINIMUM MAGNITUDE	-8.58172E+01	SIG GAIN ADDED(DB)	-1
TEST LOCATION		MINIMUM PHASE	-1.74613E+02	SIG DELAY ADDED(NS)	-1
TEST POINT		MAXIMUM FREQUENCY	9.82179E+06	REF PROBE ID	
TEST TYPE	TEST	MAXIMUM MAGNITUDE	-3.18891E+01	REF GAIN ADDED(DB)	-1
TEST DESCRIPTION	TEST WAVE	MAXIMUM PHASE	-2.94317E-01	REF DELAY ADDED(NS)	-1
TEST ENGINEER	OPERATOR	PARSEVAL TIME VALUE	4.15402E+02	INPUT WAVEFORM ID	
TEST ELEMENT		PARSEVAL FREQ VALUE	4.16811E+02	INPUT WAVEFORM SCALE	0.000000E-01
NET ANAL DISP REF	-1	PARSEVAL RATIO	-1.000000E-03	MULTI/SINGLE	
LOG ID		TIME DOMAIN DELTA-T		TAPE FILE ID	
PLOT FORMAT		T.F. CAL FILE ID		R.F. CAL FILE ID	
PHASE UNWRAP DELAY					
INPUT FILE #1	DEXP170.TCA	INPUT FILE #2		FUNCTION CODE/DATE	FTR28-JUN-82

TEST COMMENTS TRANSFORM OF DOUBLD EXPONENTIAL
FILES PLOTTED DEXP170.FCA DEXP170.TCA

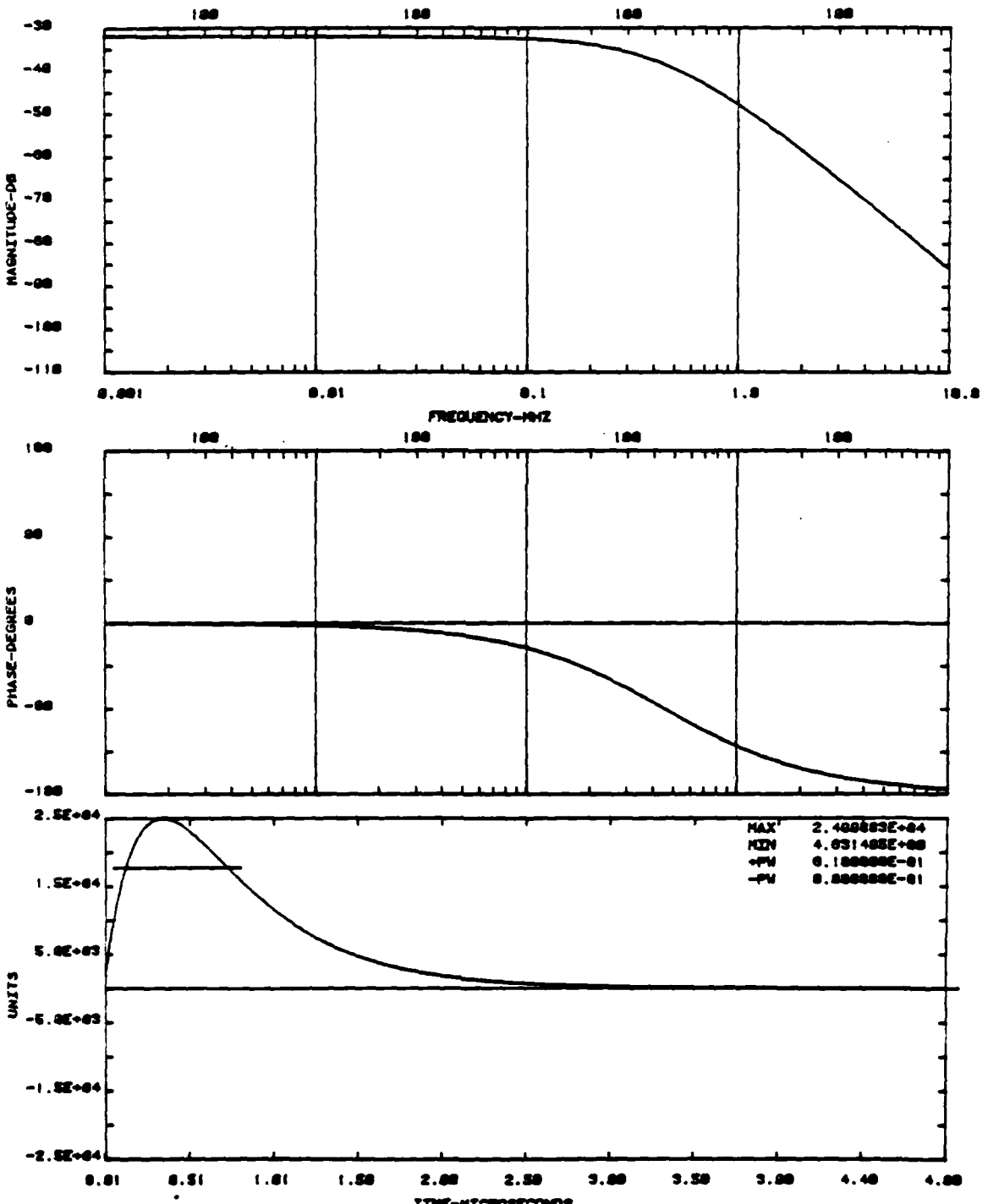


Figure I-1

17:58:38	UN-02	MINIMUM FREQUENCY	1.00000E+03	SIGNAL PROBE ID
TEST SEQL	E 0	MINIMUM MAGNITUDE	-8.58178E+01	SIG GAIN ADDED(DB)
TEST LOCA	N	MINIMUM PHASE	-1.74435E+02	SIG DELAY ADDED(NS)
TEST POIN		MAXIMUM FREQUENCY	9.82179E+06	REF PROBE ID
TEST TYPE		MAXIMUM MAGNITUDE	-3.28413E+01	REF GAIN ADDED(DB)
TEST DESC	TION	MAXIMUM PHASE	-2.69999E-01	REF DELAY ADDED(NS)
TEST ENGI	R	PARSEVAL TIME VALUE	1.37835E+00	INPUT WAVEFORM ID
TEST ELEM		PARSEVAL FREQ VALUE	1.40288E+00	INPUT WAVEFORM SCALE
NET ANAL	P REF	PARSEVAL RATIO	9.99999E-03	MULTI/SINGLE
LOG ID		TIME DOMAIN DELTA-T		TAPE FILE ID
PLOT FORM		T.F. CAL FILE ID		R.F. CAL FILE ID
PHASE UNW	DELAY			
INPUT FIL	1	INPUT FILE 02		FUNCTION CODE/DATE 16-MAR-01
TEST COMM	S	HAND CALCULATED TRANSFORM		
FILES PLO	D	HANDC000.FCA		

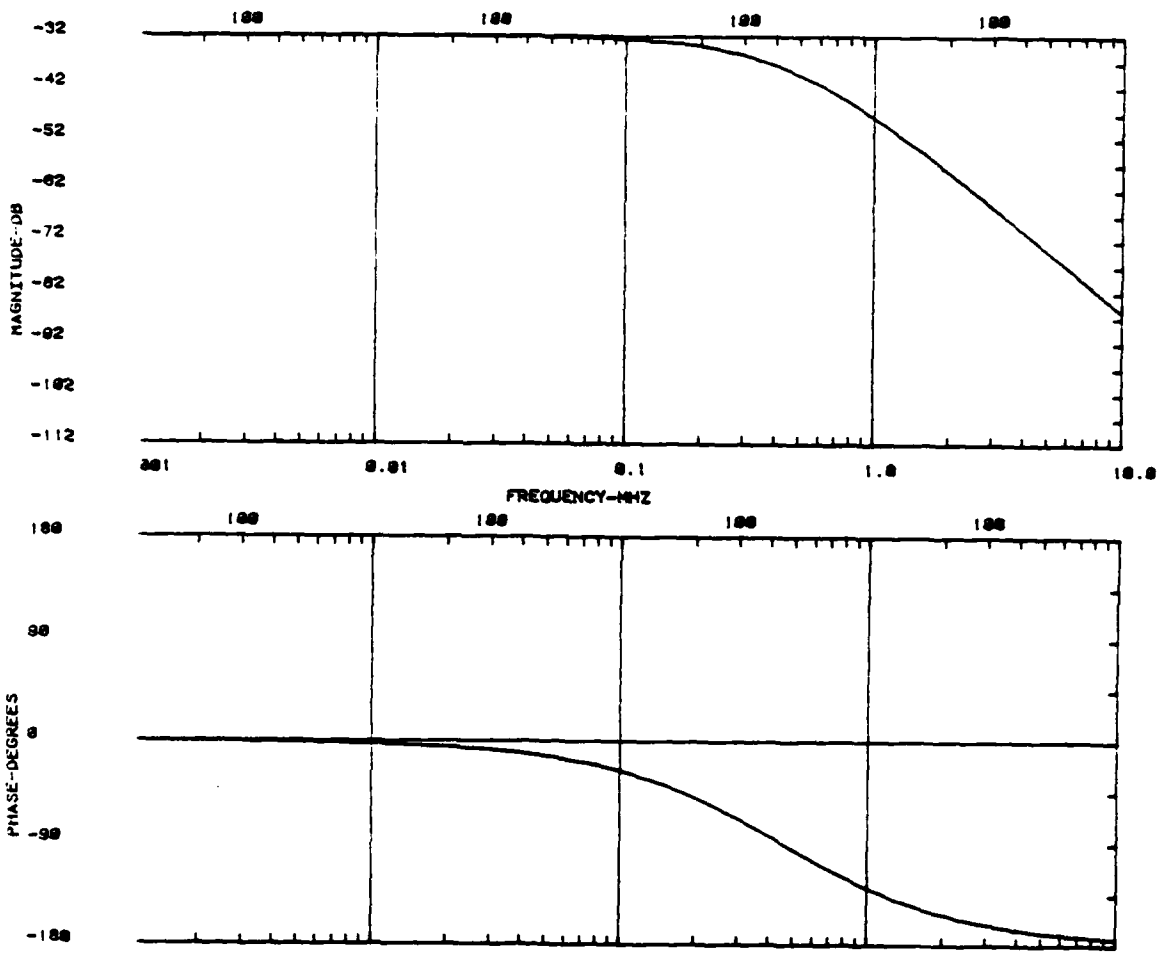


Figure I-2

10:22:59 28-JUN-82		MINIMUM FREQUENCY	1.00000E+03	SIGNAL PROBE ID	
TEST SEQUENCE #	1234	MINIMUM MAGNITUDE	-8.56172E+01	SIG GAIN ADDED(DB)	-1
TEST LOCATION		MINIMUM PHASE	-1.74613E+02	SIG DELAY ADDED(NS)	-1
TEST POINT		MAXIMUM FREQUENCY	9.82179E+06	REF PROBE ID	
TEST TYPE	TEST	MAXIMUM MAGNITUDE	-3.18991E+01	REF GAIN ADDED(DB)	-1
TEST DESCRIPTION	TEST WAVE	MAXIMUM PHASE	-2.84317E-01	REF DELAY ADDED(NS)	-1
TEST ENGINEER	OPERATOR	PARSEVAL TIME VALUE	4.15402E+02	INPUT WAVEFORM ID	
TEST ELEMENT		PARSEVAL FREQ VALUE	4.16811E+02	INPUT WAVEFORM SCALE	0.00000E-01
NET ANAL DISP REF	-1	PARSEVAL RATIO	-1.00000E-03	MULTI/SINGLE	
LOG ID		TIME DOMAIN DELTA-T		TAPE FILE ID	
PLOT FORMAT		T.F. CAL FILE ID		R.F. CAL FILE ID	
PHASE UNWRAP DELAY					
INPUT FILE #1	DEXP179.TCA	INPUT FILE #2		FUNCTION CODE/DATE	FTR28-JUN-82
TEST COMMENTS	TRANSFORM OF DOUBLD EXPONENTIAL				
FILES PLOTTED	DEXP179.FCA HANDC869.FCA				

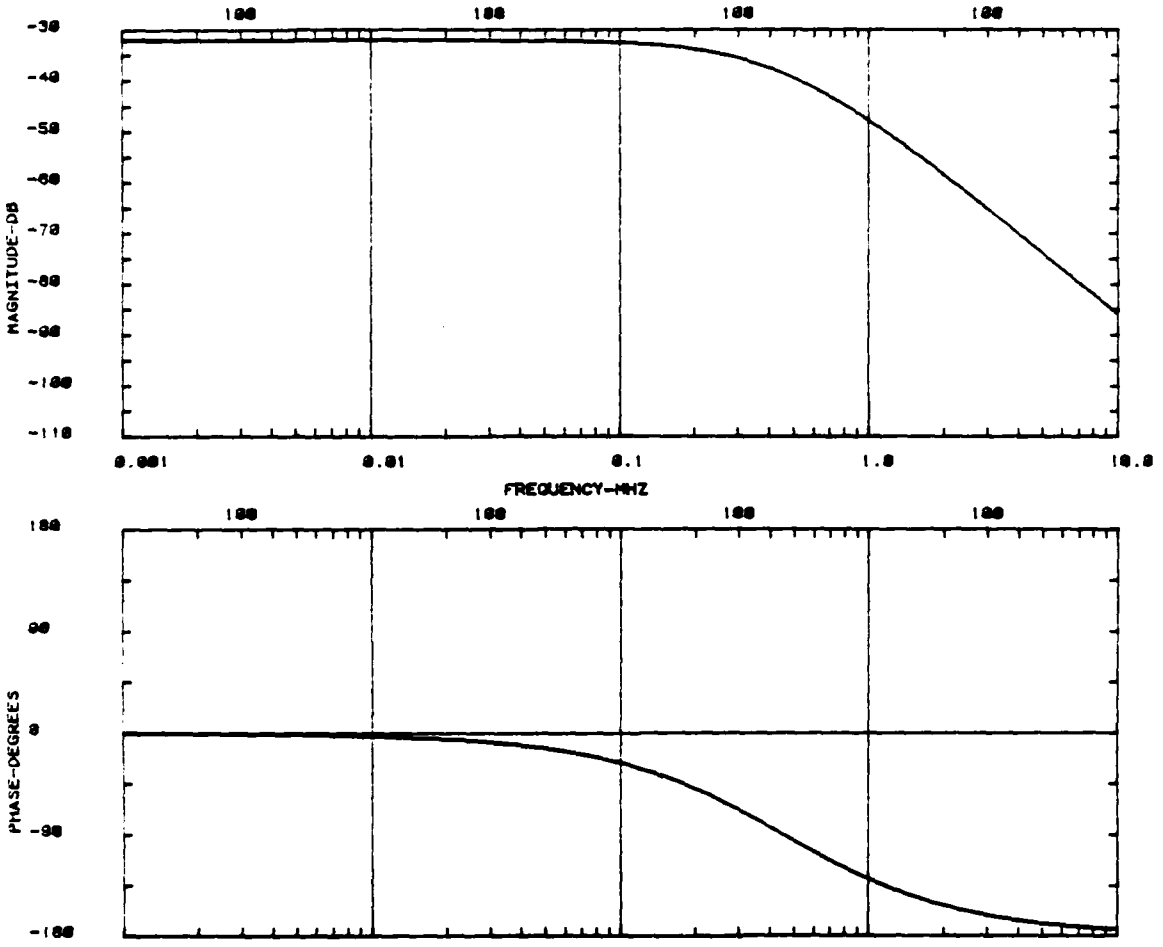


Figure I-3

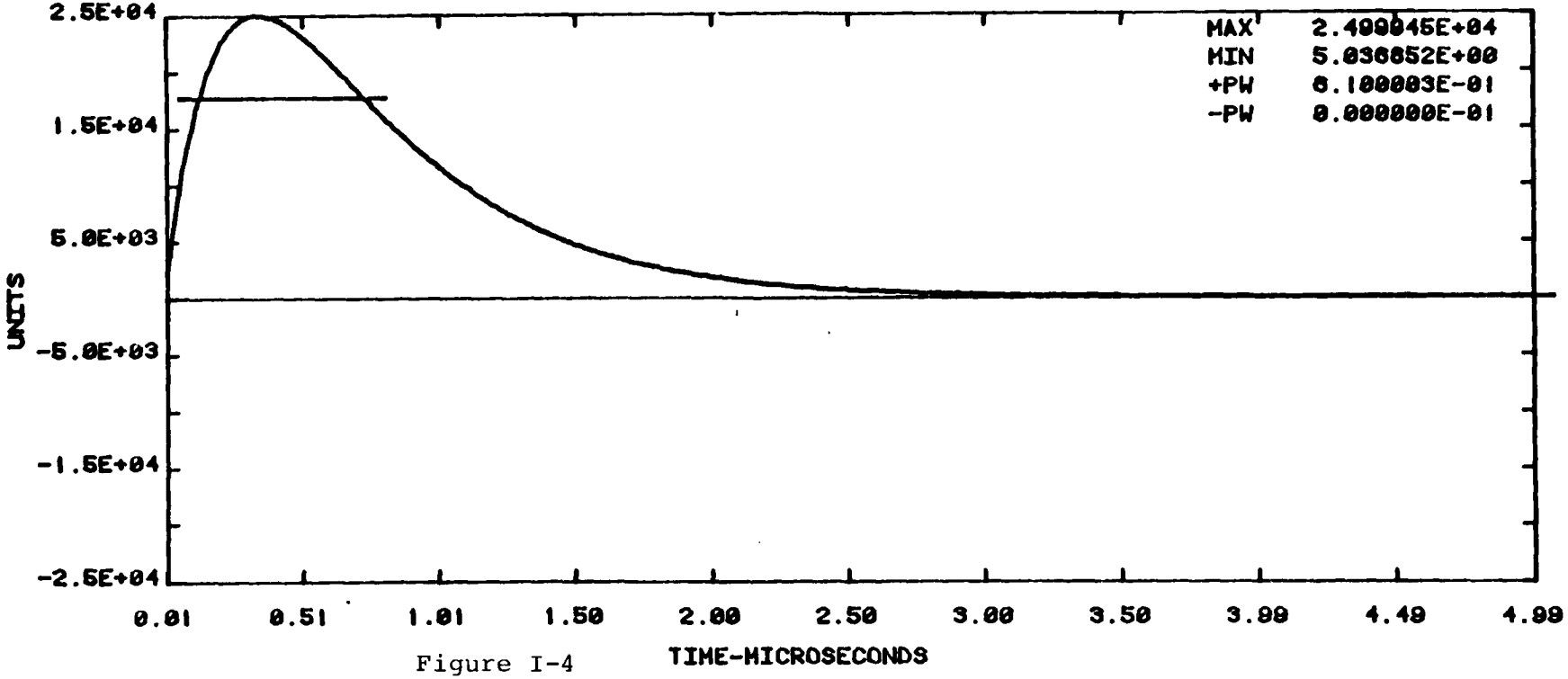

2.5E+04
1.5E+04
5.0E+03
-5.0E+03
-1.5E+04
-2.5E+04
UNITS
MAX 2.499045E+04
MIN 5.036852E+00
+PW 6.100003E-01
-PW 0.000000E-01
0.01
0.51
1.01
1.50
2.00
2.50
3.00
3.50
3.99
4.49
4.99
TIME-MICROSECONDS

Figure I-4

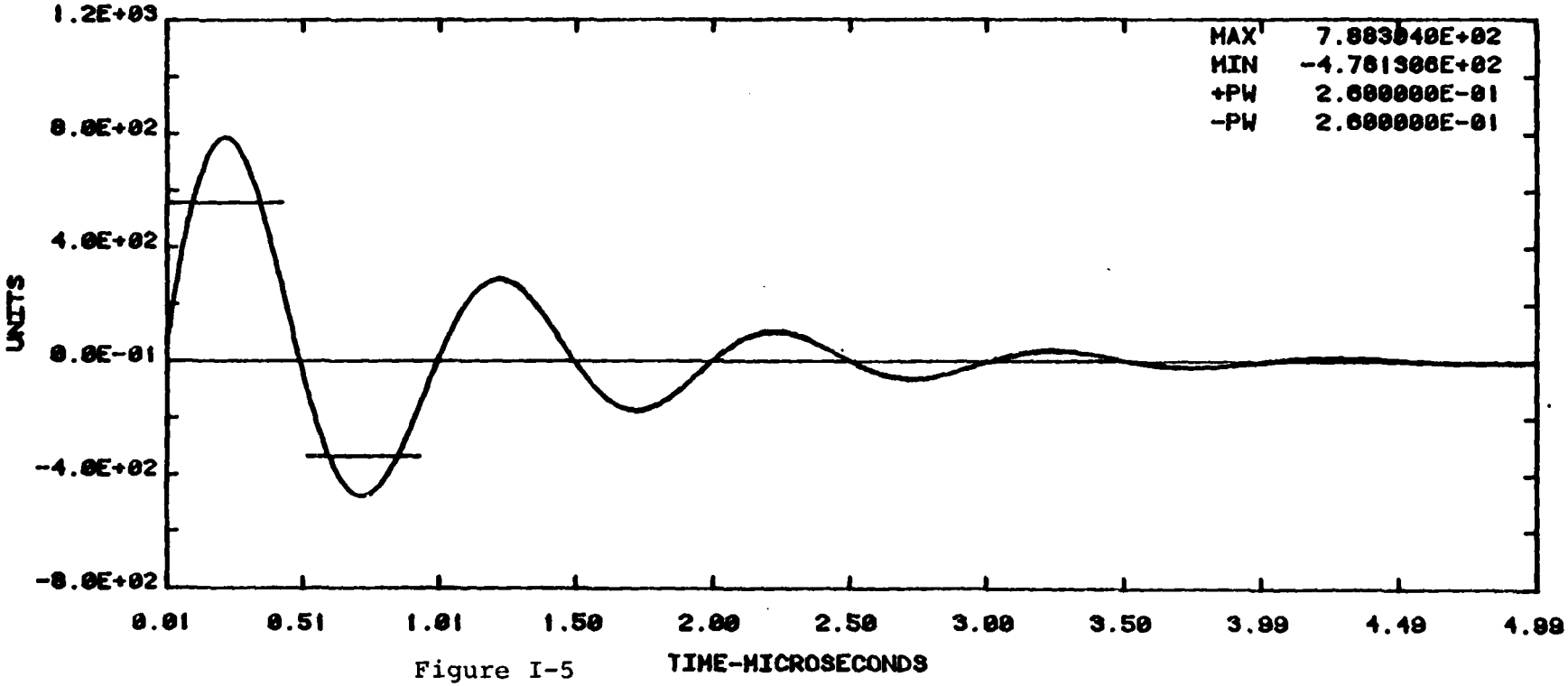

MAX 7.883040E+02
MIN -4.781306E+02
+PW 2.600000E-01
-PW 2.600000E-01
1.2E+03
8.0E+02
4.0E+02
0.0E-01
-4.0E+02
-8.0E+02
UNITS
0.01
0.51
1.01
1.50
2.00
2.50
3.00
3.50
3.99
4.49
4.99
TIME-MICROSECONDS

Figure I-5

09:13:27 01-JUN-82		MINIMUM TIME	0.00000E-01	SIGNAL PROBE ID	
TEST SEQUENCE #	0001	MINIMUM AMPLITUDE	-1.42514E+01	SIG GAIN ADDED(DB)	-1
TEST LOCATION				SIG DELAY ADDED(NS)	-1
TEST POINT		MAXIMUM TIME	4.00023E-06	REF PROBE ID	
TEST TYPE		MAXIMUM AMPLITUDE	1.13555E+02	REF GAIN ADDED(DB)	
TEST DESCRIPTION	TEST WAVE			REF DELAY ADDED(NS)	
TEST ENGINEER	OPERATOR	PARSEVAL TIME VALUE		INPUT WAVEFORM ID	
TEST ELEMENT		PARSEVAL FREQ VALUE		INPUT WAVEFORM SCALE	0.00000E-01
NET ANAL DISP REF		PARSEVAL RATIO		MULTI/SINGLE	
LOG ID		TIME DOMAIN DELTA-T	-1	TAPE FILE ID	
PLOT FORMAT		T.F. CAL FILE ID		R.F. CAL FILE ID	
PHASE UNWRAP DELAY					
INPUT FILE #1	WZERO166.TCA	INPUT FILE #2		FUNCTION CODE/DATE	DED15-JUN-82
TEST COMMENTS					
FILES PLOTTED	WZERO166.TDA				

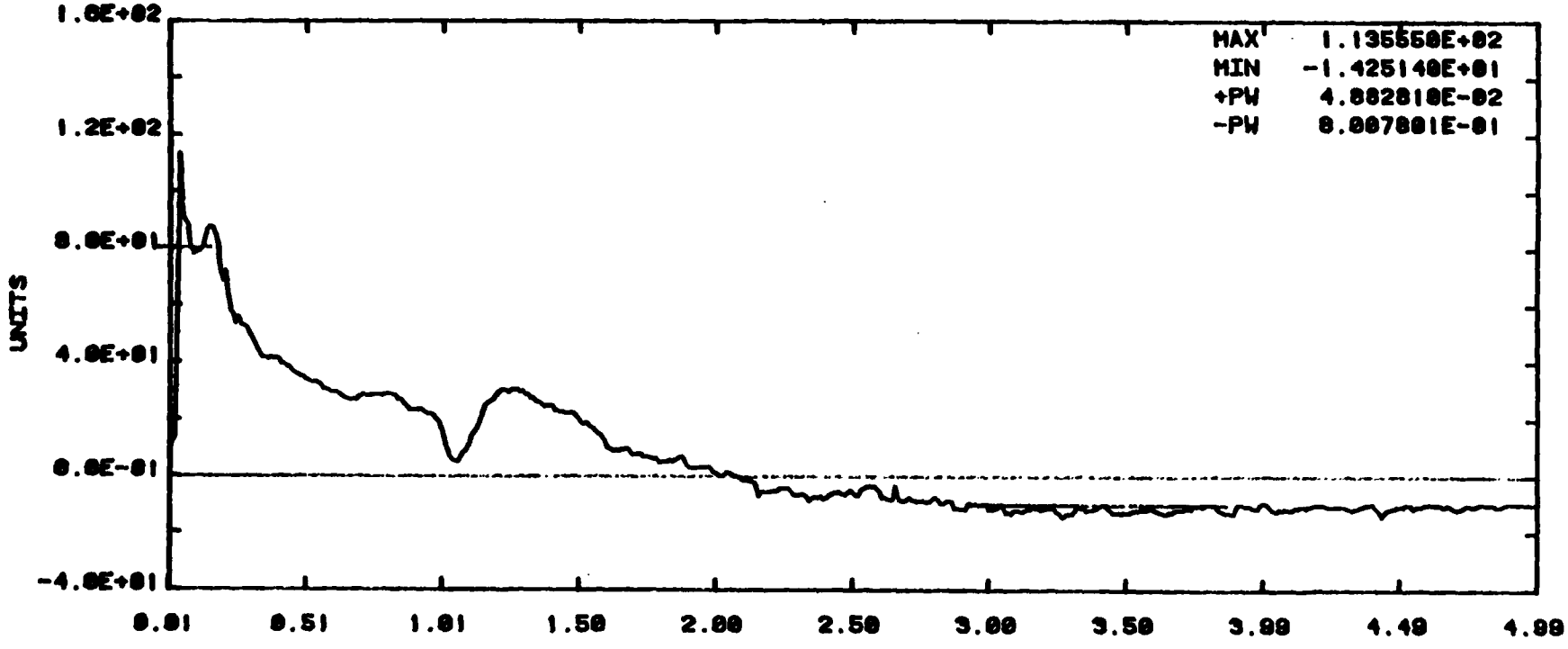


Figure I-6

00:20:1001-JUN-82		MINIMUM TIME	0.00000E-01	SIGNAL PROBE ID	
TEST SEQUENCE #	0001	MINIMUM AMPLITUDE	-1.32314E+01	SIG GAIN ADDED(DB)	-1
TEST LOCATION				SIG DELAY ADDED(NS)	-1
TEST POINT		MAXIMUM TIME	4.98997E-06	REF PROBE ID	
TEST TYPE		MAXIMUM AMPLITUDE	1.11463E+02	REF GAIN ADDED(DB)	
TEST DESCRIPTION	TEST WAVE			REF DELAY ADDED(NS)	
TEST ENGINEER	OPERATOR	PARSEVAL TIME VALUE	2.84207E-03	INPUT WAVEFORM ID	
TEST ELEMENT		PARSEVAL FREQ VALUE	4.18725E-03	INPUT WAVEFORM SCALE	0.00000E-01
NET ANAL DISP REF		PARSEVAL RATIO	1.01000E-01	MULTI/SINGLE	
LOG ID		TIME DOMAIN DELTA-T	1.00000E-08	TAPE FILE ID	
PLOT FORMAT		T.F. CAL FILE ID		R.F. CAL FILE ID	
PHASE UNWRAP DELAY					
INPUT FILE #1	WZERO174.FCA	INPUT FILE #2		FUNCTION CODE/DATE	ITR24-JUN-82

TEST COMMENTS	TRANSFORMED THREAT
FILES PLOTTED	WZTRA175.TCA

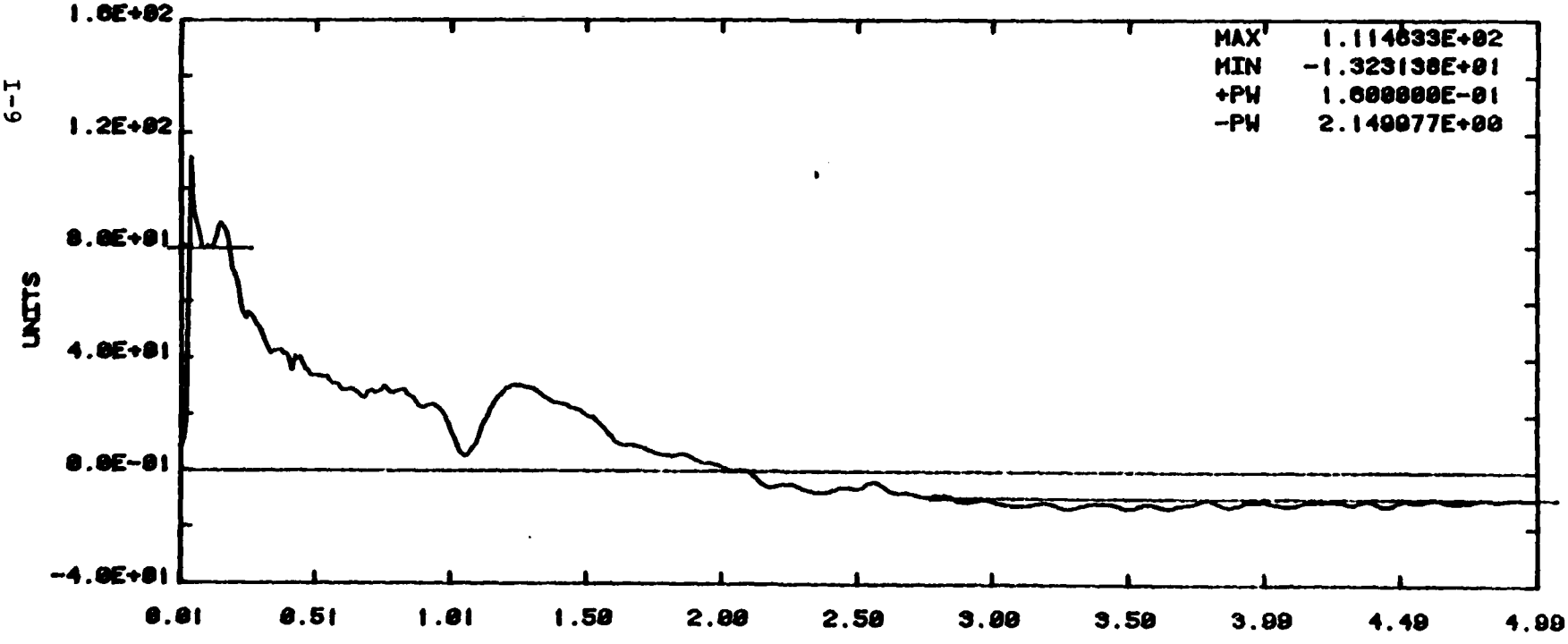


Figure I-7

09:13:27 01-JUN-82		MINIMUM TIME	0.000000E-01	SIGNAL PROBE ID	
TEST SEQUENCE #	0001	MINIMUM AMPLITUDE	-1.425146+01	SIG GAIN ADDED(DB)	-1
TEST LOCATION				SIG DELAY ADDED(NS)	-1
TEST POINT		MAXIMUM TIME	4.00023E-06	REF PROBE ID	
TEST TYPE		MAXIMUM AMPLITUDE	1.13555E+02	REF GAIN ADDED(DB)	
TEST DESCRIPTION	TEST WAVE			REF DELAY ADDED(NS)	
TEST ENGINEER	OPERATOR	PARSEVAL TIME VALUE		INPUT WAVEFORM ID	
TEST ELEMENT		PARSEVAL FREQ VALUE		INPUT WAVEFORM SCALE	0.000000E-01
NET ANAL DISP REF		PARSEVAL RATIO		MULTI/SINGLE	
LOG ID		TIME DOMAIN DELTA-T	-1	TAPE FILE ID	
PLOT FORMAT		T.F. CAL FILE ID		R.F. CAL FILE ID	
PHASE UNWRAP DELAY					
INPUT FILE #1	WZERO166.TCA	INPUT FILE #2		FUNCTION CODE/DATE	DED15-JUN-82

TEST COMMENTS
FILES PLOTTED WZERO166.TDA WZTRA175.TCA

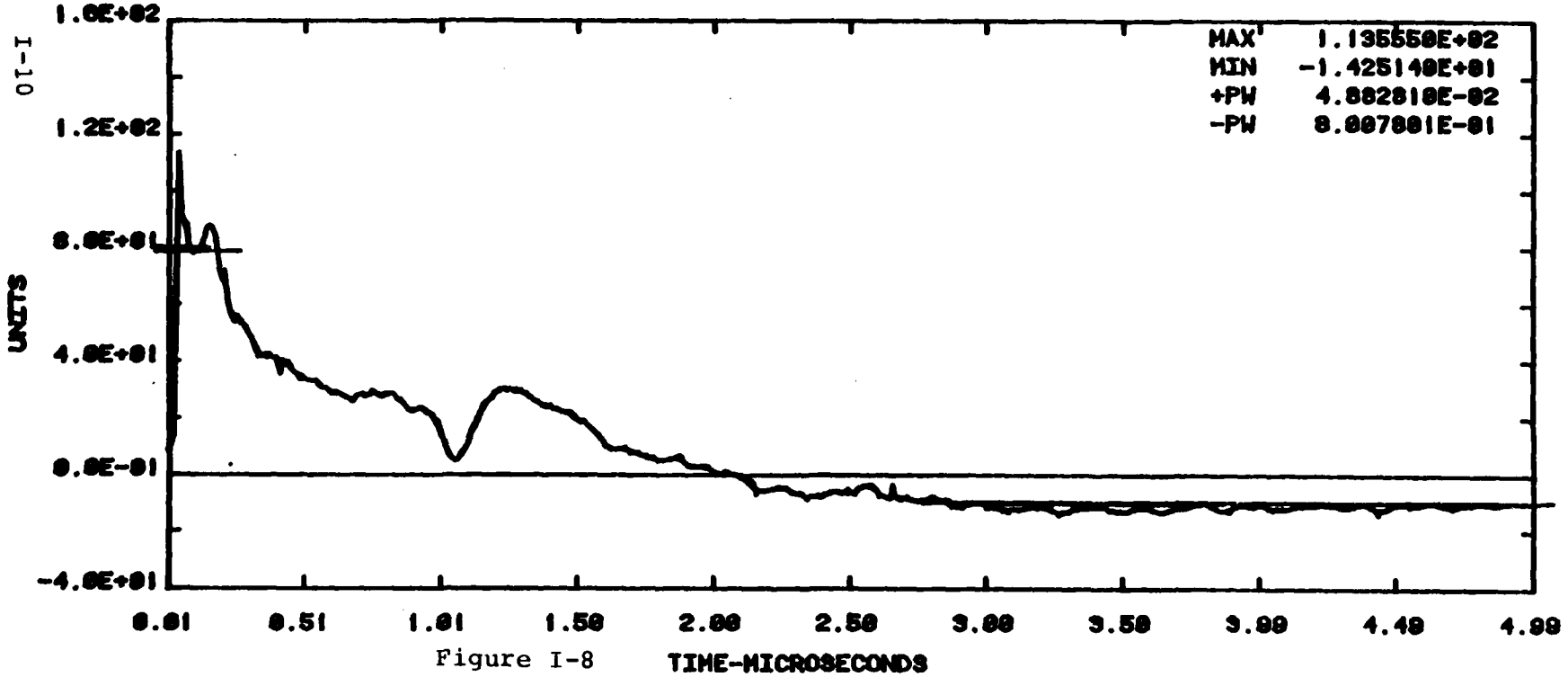


Figure I-8

10:10:17 10-JUN-92		MINIMUM FREQUENCY	1.000000E+04	SIGNAL PROBE ID	TSTPR
TEST SEQUENCE #	1111	MINIMUM MAGNITUDE	-1.578000E+01	SIG GAIN ADDED(DB)	-40
TEST LOCATION	ALAMO	MINIMUM PHASE	-1.759000E+02	SIG DELAY ADDED(NS)	0
TEST POINT	EGG CP TEST	MAXIMUM FREQUENCY	9.772000E+07	REF PROBE ID	TEST2
TEST TYPE	TEST	MAXIMUM MAGNITUDE	2.436000E+01	REF GAIN ADDED(DB)	-18
TEST DESCRIPTION	SMARTTEST	MAXIMUM PHASE	1.762000E+02	REF DELAY ADDED(NS)	0
TEST ENGINEER	DAVEY CROCKE	PARSEVAL TIME VALUE		INPUT WAVEFORM ID	BUTTERWTH
TEST ELEMENT	PATIENCE	PARSEVAL FREQ VALUE		INPUT WAVEFORM SCALE	1.
NET ANAL DISP REF	-40	PARSEVAL RATIO		MULTI/SINGLE	SINGLE
LOG ID	STARDATE 1.1	TIME DOMAIN DELTA-T		TAPE FILE ID	8188T1111169
PLOT FORMAT	TFA	T.F. CAL FILE ID	8188	R.F. CAL FILE ID	8188
PHASE UNWRAP DELAY					
INPUT FILE #1		INPUT FILE #2		FUNCTION CODE/DATE	ONL10-JUN-92
TEST COMMENTS	TEST				
FILES PLOTTED	T1111169.CMA	T1111169.TCA			

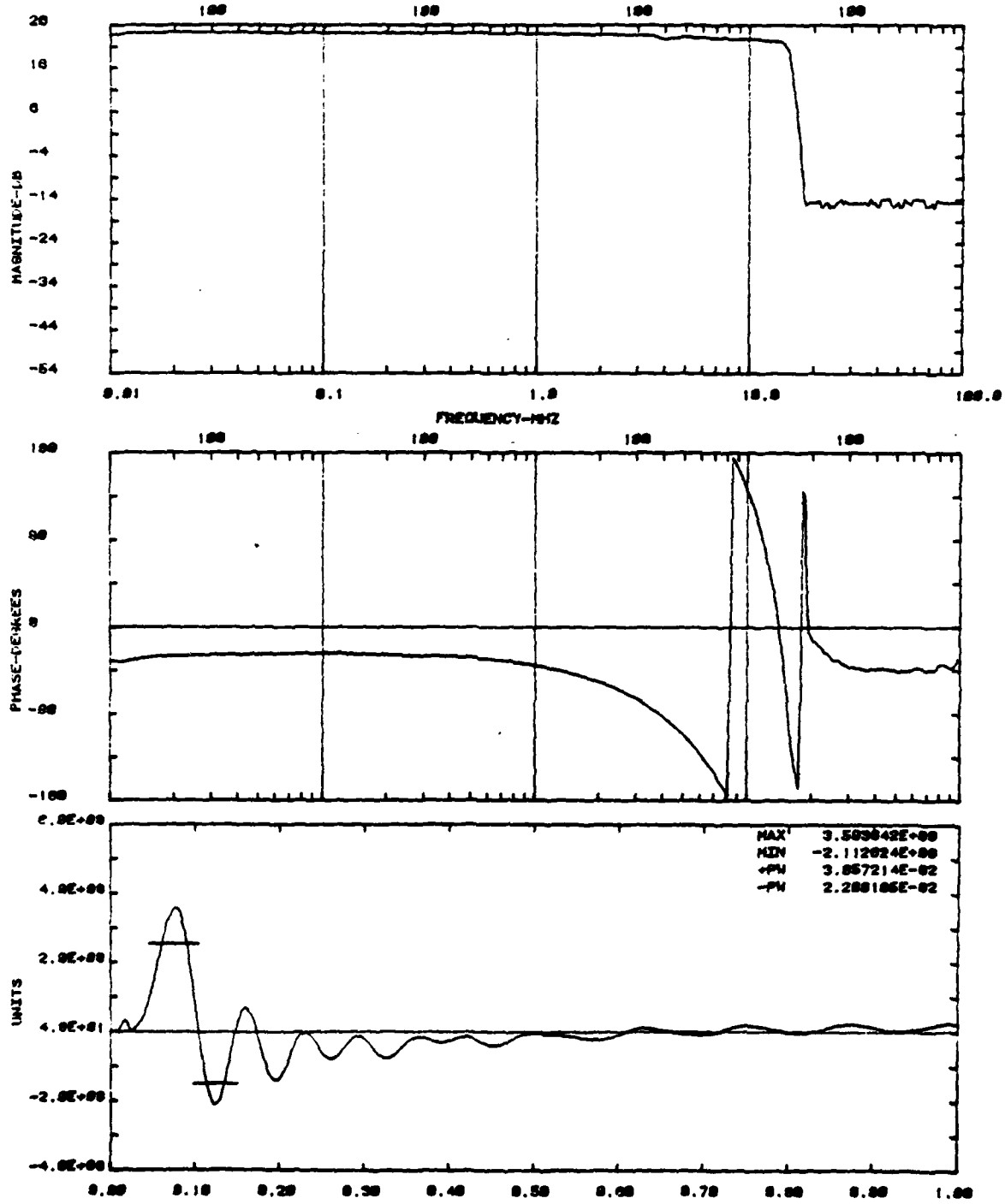


Figure I-9

10:10:17 10-JUN-92		MINIMUM FREQUENCY	1.000000E+04	SIGNAL PROBE ID	TSTPR
TEST SEQUENCE #	1111	MINIMUM MAGNITUDE	-2.197900E+02	SIG GAIN ADDED(DB)	-40
TEST LOCATION	ALAMO	MINIMUM PHASE	-1.775290E+02	SIG DELAY ADDED(NS)	0
TEST POINT	ESB CP TEST	MAXIMUM FREQUENCY	9.772900E+07	REF PROBE ID	TEST2
TEST TYPE	TEST	MAXIMUM MAGNITUDE	2.421100E+01	REF GAIN ADDED(DB)	-10
TEST DESCRIPTION	SMARTTEST	MAXIMUM PHASE	1.798200E+02	REF DELAY ADDED(NS)	0
TEST ENGINEER	DAVEY CROCKE	PARSEVAL TIME VALUE		INPUT WAVEFORM ID	BUTTERWTH
TEST ELEMENT	PATIENCE	PARSEVAL FREQ VALUE		INPUT WAVEFORM SCALE	1.
NET ANAL DISP REF	-40	PARSEVAL RATIO		MULTI/SINGLE	SINGLE
LOG ID	STARDATE 1.1	TIME DOMAIN DELTA-T		TAPE FILE ID	S1BBT1111169
PLOT FORMAT	TFA	T.F. CAL FILE ID	S1BB	R.F. CAL FILE ID	S1BB
PHASE UNWRAP DELAY					
INPUT FILE #1	T1111169.CHA	INPUT FILE #2	BUTER167.FDA	FUNCTION CODE/DATE	MUL24-JUN-92
TEST COMMENTS	TEST DATE = BUTTERWORTH				
FILES PLOTTED	T11BT178.FCA	T11BT178.TCA			

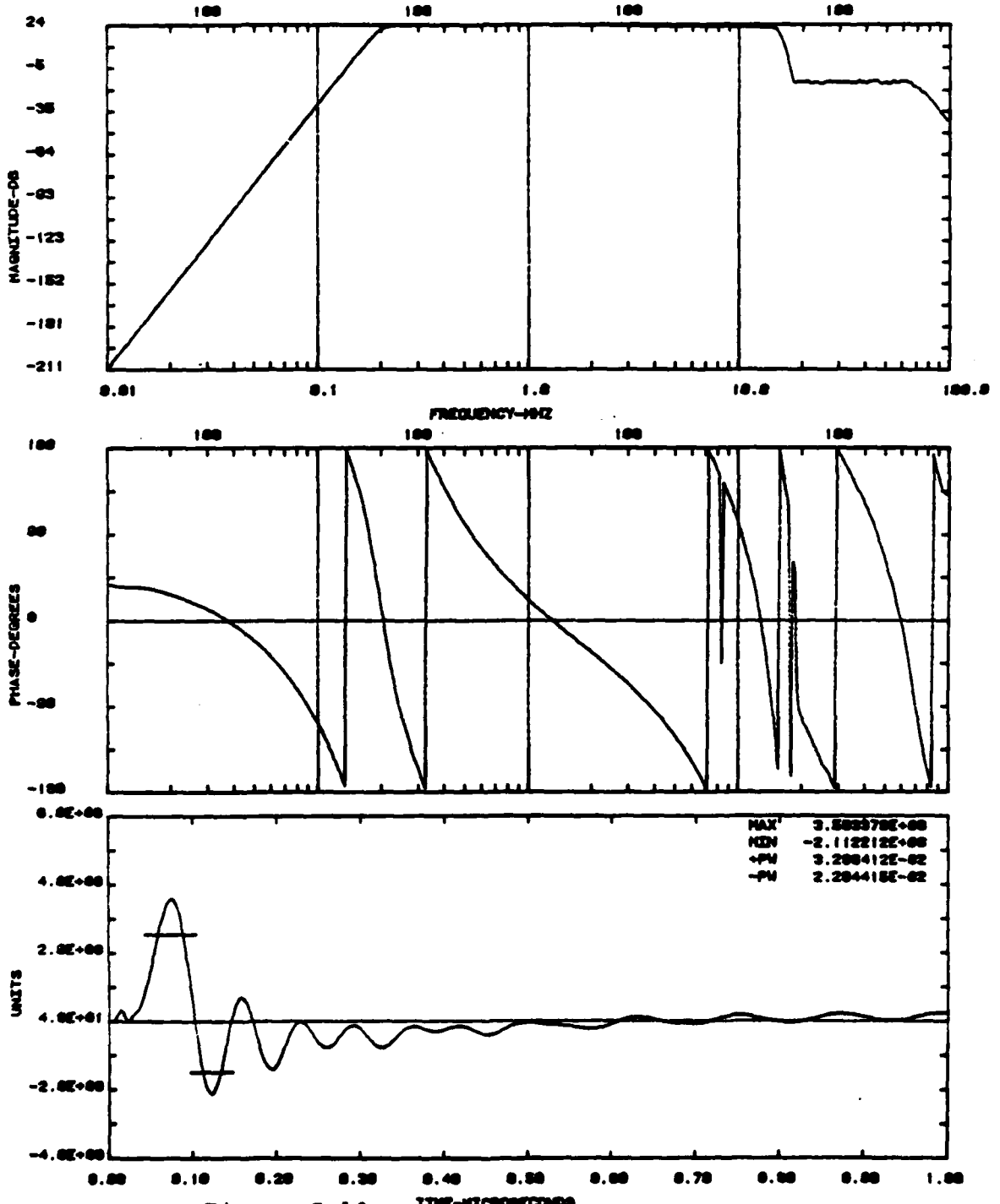


Figure I-10

10:10:17 18-JUN-82		MINIMUM TIME	0.00000E-01	SIGNAL PROBE ID	TSTPR
TEST SEQUENCE #	1111	MINIMUM AMPLITUDE	-2.112626+00	SIG GAIN ADDED(DB)	-40
TEST LOCATION	ALAMO			SIG DELAY ADDED(NS)	0
TEST POINT	E88 CP TEST	MAXIMUM TIME	1.00000E-06	REF PROBE ID	TEST2
TEST TYPE	TEST	MAXIMUM AMPLITUDE	3.59364E+00	REF GAIN ADDED(DB)	-10
TEST DESCRIPTION	SMARTTEST			REF DELAY ADDED(NS)	0
TEST ENGINEER	DAVEY CROCKE	PARSEVAL TIME VALUE	5.93620E+00	INPUT WAVEFORM ID	BUTTERWTH
TEST ELEMENT	PATIENCE	PARSEVAL FREQ VALUE	6.15665E+00	INPUT WAVEFORM SCALE	1.
NET ANAL DISP REF	-40	PARSEVAL RATIO	1.00000E-02	MULTI/SINGLE	SINGLE
LOG ID	STARDATE 1.1	TIME DOMAIN DELTA-T	-1. .	TAPE FILE ID	0100T1111169
PLOT FORMAT	TFA	T.F. CAL FILE ID	0100	R.F. CAL FILE ID	0100
PHASE UNWRAP DELAY					
INPUT FILE #1		INPUT FILE #2		FUNCTION CODE/DATE	ONL18-JUN-82
TEST COMMENTS	TEST				
FILES PLOTTED	T1111169.TCA	T11BT175.TCA			

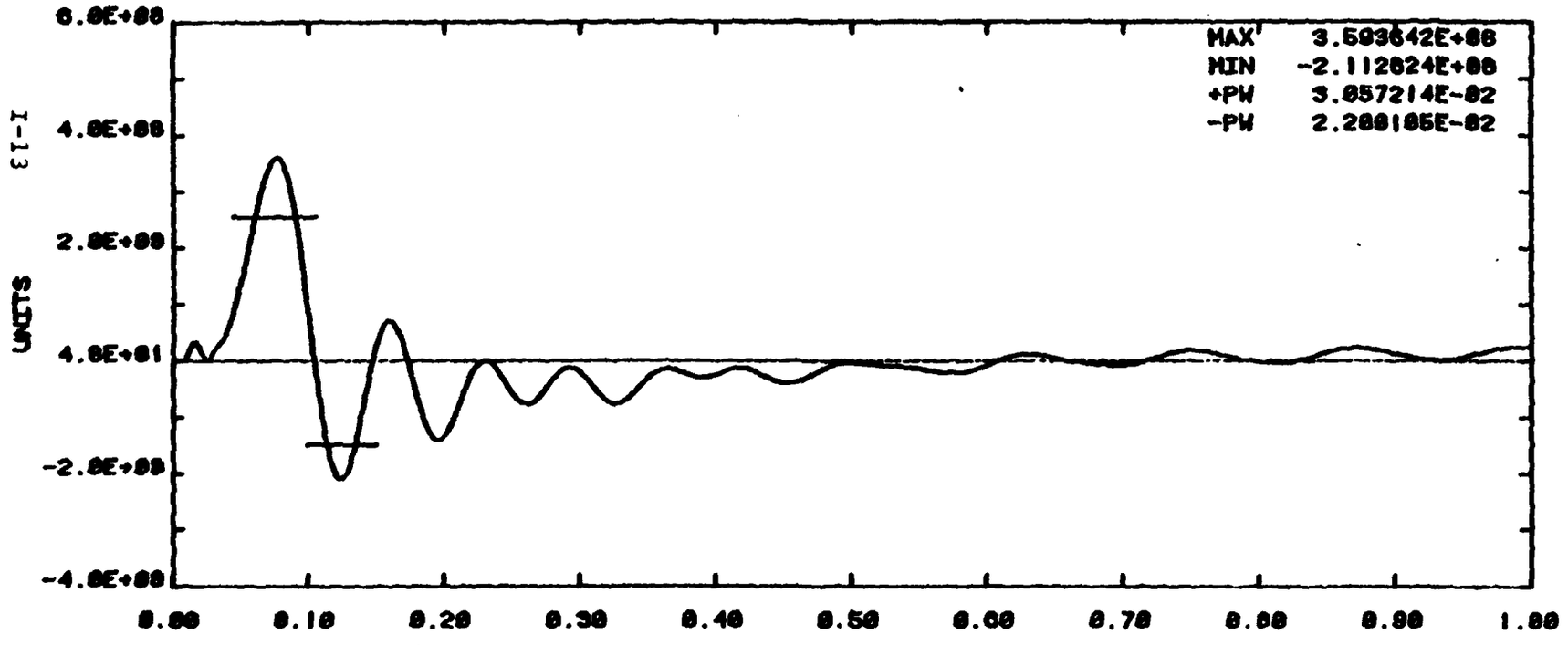


Figure I-11

DISTRIBUTION LIST

DEPARTMENT OF DEFENSE

Defense Nuclear Agency
ATTN: NATA
ATTN: RAEE
ATTN: RAAE
4 cys ATTN: STTI/CA

Defense Tech Info Center
12 cys ATTN: DD

Field Command, Defense Nuclear Agency
ATTN: FCTT, W. Summa
ATTN: FCTXE
ATTN: FCPR

National Communications System
ATTN: NCS-TS

Harry Diamond Laboratories
ATTN: Chief 21500
ATTN: DRDEL-CT
ATTN: 00100 Commander/Tech Dir/Div Dir
ATTN: DELHD-NW, J. Bombardt, 20000
ATTN: Chief Div 10000
ATTN: Chief Div 40000
ATTN: Chief Div 50000
ATTN: DELHD-TA-L
ATTN: Chief Div 20000
ATTN: Chief Div 30000
2 cys ATTN: Chief 21000
2 cys ATTN: Chief 22000
3 cys ATTN: Chief 20240

DEPARTMENT OF THE ARMY

US Army Nuclear & Chemical Agency
ATTN: Library

DEPARTMENT OF THE NAVY

Naval Ocean Systems Center
ATTN: Code 4471

Naval Shore Elect & Engrg Actvy, Pacific
ATTN: D. Koide

Naval Surface Weapons Center
ATTN: Code F32
ATTN: Code F30

DEPARTMENT OF THE AIR FORCE

Air Force Weapons Laboratory
ATTN: SUL

DEPARTMENT OF DEFENSE CONTRACTORS

American Telephone & Telegraph Co
ATTN: W. Edwards

BDM Corp
ATTN: W. Sweeney
ATTN: L. Jacobs

Boeing Co
ATTN: R. Scheppe, MS 9F-01

TRW, Inc
ATTN: R. Hendrickson

DEPARTMENT OF DEFENSE CONTRACTORS (Continued)

Booz-Allen & Hamilton, Inc
ATTN: D. Durgin

EG&G Wash Analytical Svcs Ctr, Inc
ATTN: A. Bonham
2 cys ATTN: J. Bridges
2 cys ATTN: J. Burns, IV
2 cys ATTN: G. Gagliano
2 cys ATTN: R. Reinman
2 cys ATTN: R. Nelson
2 cys ATTN: P. Lindsey

Georgia Institute of Technology
ATTN: Res & Sec Coord for H. Denny

GTE Communications Products Corp
ATTN: R. Steinhoff

Horizons Technology, Inc
ATTN: R. Lewis

IRT Corp
ATTN: R. Stewart
ATTN: B. Williams

JAYCOR
ATTN: R. Poll
ATTN: R. Schaefer

Kaman Tempo
ATTN: DASIAC

Kaman Tempo
ATTN: DASIAC

Mission Research Corp
ATTN: W. Ware
ATTN: J. Lubell

Pacific-Sierra Research Corp
ATTN: H. Brode, Chairman SAGE

R&D Associates
ATTN: P. Haas
ATTN: W. Karzas
ATTN: W. Graham

R&D Associates
ATTN: Library

Rockwell International Corp
ATTN: G. Morgan

Science & Engrg Associates, Inc
ATTN: V. Jones

SRI International
ATTN: A. Whitson
ATTN: E. Vance

Systems Research & Applications Corp
ATTN: S. Greenstein

TRW Electronics & Defense Sector
ATTN: J. Brossier

TRW Electronics & Defense Sector
ATTN: Librarian

END

FILMED

4-85

DTIC

END

FILMED

4-85

DTIC